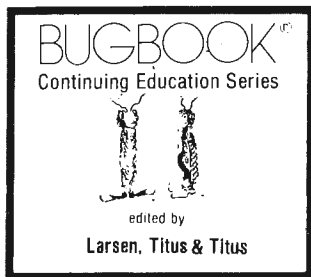
DBUG

Un programma interprete per la messa a punto del software 8080

EDIZIONE ITALIANA

C.A. TITUS
J.A. TITUS

JACKSON ITALIANA EDITRICE

DBUG

Un programma interprete per la messa a punto del software 8080

di
Cristopher A. Titus
Tychon Inc
Blacksburg, Virginia 24060

e

Jonathan A. Titus
Tychon Inc
Blacksburg, Virginia 24060

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12 - 20124 Milano

PREFAZIONE ALL'EDIZIONE ITALIANA

L'utente di sistemi a microprocessore trova sempre più diffusi, nel campo della letteratura specifica, esempi di applicazioni hardware che gli consentono di risolvere facilmente i propri problemi di interfacciamento. Altrettanto non può dirsi per quanto riguarda la produzione di strumenti software che aiutino l'utente nella stesura dei programmi.

Mano a mano che sistemi a microprocessore si vanno diffondendo è quindi sempre più sentita, da parte dell'utente, l'esigenza di disponibilità di librerie software a cui attingere di volta in volta i supporti necessari alla programmazione.

Pensiamo perciò che questo libro riscuota l'interesse di chi oggi utilizza sistemi a microprocessore. Esso è altresì interessante in quanto è sviluppato sull'8080, ancora oggi il più diffuso dei microprocessori, e rappresenta un approfondimento sull'operatività dell'8080 come CPU di un sistema.

Nel presentare questo testo, che consideriamo un interessante contributo allo sviluppo della produzione di software, ci auguriamo che questo sviluppo non proceda a senso unico "produttore-utente", ma che si creino nuove e utili possibilità di scambi tra utenti e utenti.

Infine una nota di carattere tecnico per coloro che già possedessero un microcomputer MMD1 corredato di scheda di espansione e interfaccia, MMD1 MI. È possibile un utilizzo immediato del programma di DBUG (disponibile su quattro EPROM presso i maggiori rivenditori di componenti) in quanto le sobrutines di I/O sono compatibili con gli indirizzi di dispositivo assegnati agli UART's disposti sulla scheda di espansione e interfaccia.

MICROLEM divisione didattica

PREFAZIONE DEGLI AUTORI

È questo il primo testo, della serie Bugbook Application, che presenta un programma per microcomputer. Ci auguriamo che troverete il problema DBUG utile allo sviluppo del software 8080, così come altri hanno riscontrato l'utilita degli altri Bugbook nello sviluppo dell'hardware digitale e delle tecniche di interfacciamento dei microcomputer. Abbiamo scritto il DBUG perchè può facilitare il vostro lavoro di programmazione. Con DBUG, potete inserire e cambiare i dati e i passi di programma immagazzinati nella memoria di lettura/scrittura. Dopo che un programma è stato inserito, potete procedere con la tecnica del passo passo attraverso il programma stesso e notare l'effetto che una particolare istruzione ha su ognuno dei registri dell'8080, sulla locazione di memoria indirizzata dai registri H e L, sullo stack pointer e sugli ultimi due valori caricati nello stack. Al contrario della tecnica hardware del passo passo, in cui l'8080 procede di solito attraverso cicli di istruzioni individuali, DBUG procede attraverso un'istruzione completa, a prescindere dal numero di cicli richiesti. DBUG è inoltre in grado di leggere e perforare un nastro di carta usando una telescrivente. Una volta che un programma funziona in modo corretto, potete sfruttare questa caratteristica per conservare il programma stesso su di un nastro perforato.

Potete anche conservare su nastro i valori dei dati memorizzati, dato che non ha importanza, per un DBUG, che cosa viene perforato sul nastro stesso. In un secondo tempo, tali nastri possono essere memorizzati nel microcomputer 8080 usando la funzione di lettura di nastro, associata a DBUG.

DBUG, in origine, era stato scritto per essere usato con una telescrivente. Comunque, le ruotine di ingresso/uscita (I/O) possono essere facilmente cambiate ed essere utilizzate per una cassetta audio o un terminale video (CRT), che vengono entrambi spesso usati con i microcomputer 8080. Non dovreste incontrare problemi nel posizionare le routine di I/O contenute all'interno di DBUG. Infatti, la maggior parte delle volte, noi, alla Tychon, usiamo un CRT con DBUG proprio per via della sua elevata velocità di trasmissione dei dati, passando ad una telescrivente solo per perforare o leggere un nastro.

Quando abbiamo iniziato a scrivere DBUG, una delle nostre esigenze più importanti era quella di contenere il programma in 1024 (1K) locazioni di memoria, usando i dispositivi di lettura/scrittura, EPROM, PROM e ROM. Quando le EPROM 1072A erano costose, pensammo che per un sistema nascente, quattro EPROM fossero un'esigenza ragionevole. Ora che le 1072A sono facilmente disponibili e che le nuove EPROM 2078 (1K x 8) vengono messe a disposizione da molti venditori, il costo di tali dispositivi è notevolmente diminuito. Proprio per questo, stiamo lavorando a DBUG II, un metodo molto più ampio e completo per la messa a punto dei programmi. Dal momento che abbiamo limitato DBUG ad 1K byte di memoria, troverete probabilmente che DBUG manchi di alcune delle caratteristi-

che proprie di altri package software impiegati nella messa a punto dei programmi (dbug). Nonostante questo, siamo rimasti più che mai soddisfatti delle sue prestazioni.

Nelle appendici, abbiamo incluso due listing relativi a DBUG. Uno dei listing è in codice ottale e contiene le sobroutine ottali di I/O, l'altro listing è in codice esadecimale e contiene le modifiche necessarie in modo che le sobroutine di I/O siano orientate secondo il sistema esadecimale. Se programmate in codice ottale, vorrete inserire la versione ottale di DBUG nella memoria di lettura/scrittura, oppure richiedere la versione binaria su nastro di DBUG con le subroutine ottali di I/O. Inserendo in memoria la versione esadecimale di DBUG, saranno in grado di usarlo coloro che programmano in codice esadecimale. Se vi è necesario un nastro, dovreste richiedere il nastro di DBUG in codice binario, con le subroutine di I/O in codice esadecimale.

Uno dei maggiori problemi associato alla pubblicazione di programmi relativi ai microcomputer, sta nel fatto che molti possiedono dei computer che sono leggermente diversi da tutti gli altri sistemi di elaborazione. Alcuni utenti hanno molti dispositivi periferici, quali dischi, lettori e perforatori a nastro ad alta velocità,n CRT e PROM programmer mentre molti altri hanno magari solo una telescrivente.

Un'altra variabile è costituita dal numero di byte di memoria, dal tipo di memoria stessa (lettura/scrittura o PROM) e dall'indirizzo di memoria. DBUG è stato scritto in modo da poter risiedere nel 5° blocco di memoria da 1K. Il blocco di memoria da 1K in cui DBUG risiederà, ha indirizzi compresi fra 020 000 e 023 377 (esadecimali da 10 00 a 13 FF). È necessaria anche una piccola quantità di memoria di lettura/scrittura con gli indirizzi compresi fra 003 300 e 000 376 (esadecimali da 03 CO a 03 FE) e fra 000 070 e 000 072 (esadecimali fra 00 38 e 00 3A). Dovreste avere già, a questi indirizzi, la memoria di lettura/scrittura o le PROM. Sfortunatamente, non esiste una soluzione semplice al problema di questa "allocazione di memoria". Comunque, potreste memorizzare DBUG in un'altra parte della memoria (cioé 100 000 - 103 377 o 40 00 - 43 FF) e quindi procedere attraverso DBUG con la tecnica del passo passo e cambiare tutti i byte d'indirizzo alti delle istruzioni JMP, CALL, LXI, SHLD,LHLD, STA e LDA. Dopo aver memorizzato con successo DBUG nella memoria di lettura/scrittura ed aver apportato agli indirizzi tutte le modifiche necessarie, può darsi che dobbiate cambiare anche le subroutine di I/O perché rispecchino i dispositivi di I/O che avete ora nel vostro sistema.

Un altro problema che si verifica è il modo in cui memorizzare inizialmente il nastro. Abbiamo fornito un caricatore bootstrap che potete usare per immagazzinare in memoria il nastro DBUG. Può darsi che il DBUG stesso debba essere "trasferito" in un'altra parte della memoria. Lasciamo a voi il compito di cambiare gli indirizzi HI e LO delle istruzioni JMP e CALL interne al bootstrap. Può darsi che dobbiate cambiare le ruotine d'ingresso del bootstrap (chiamata READ), a seconda del dispositivo periferico che userete per leggere il nastro. Notate che READ si trova alla fine del bootstrap il che ne facilita notevolmente il cambiamento o l'ampliamento, come richiesto dal vostro sistema.

Anche se il programma e la documentazione relativi a DBUG costituiscono il primo libro sul software della serie Bugbook Application, abbiamo già in programma delle aggiunte future riguardanti alcune subroutine utili, package matematici ed il software di editor/assembler. La nostra intenzione è non solo quella di fornire un

buon software che possa essere usato con un microcomputer, ma anche quella di fornire istruzioni operative, nonché un listing ben documentato, di semplice lettura, da poter studiare autonomamente.

Al momento, il nostro interesse è concentrato sullo sviluppo dell'assembler per i sistemi 8080. Nel caso abbiate un interesse particolare, oppure un progreamma interessante non specializzato, che potrebbe essere incorporato in un formato di questo tipo, gradiremmo esserne messi a conoscenza. Fateci cortesemente sapere che cosa pensate di DBUG, se avete dei problemi o dei suggerimenti da darci per ottenere dei miglioramenti.

La Tychon, Inc. si dedica all'informazione dello scienziato, dell'ingegnere e di coloro che hanno l'hobby degli elaboratori elettronici, nel campo dell'elettronica digitale, nonché dell'hardware e del software dei microcomputer.

Attualmente, i nostri articoli appaiono mensilmente sul "*American Laboratory*" (International Scientific Communications, Inc. 8080 Kings Highway, Fairfield, CT 06430), sulla rivista tailandese "*Semiconductor Electronics Journal*" (Science, Engineering and Education Co., Ltd., Bangkok, Thailandia) e sulla rivista "*Elektroniker*" (Burgdorf, Switzerland). Appaiono anche nostri articoli su "*Electronic News*" (IPC Business Press Pty Ltd., Australia), su "*Ham Radio*" (Greenville, NH), su "*Computer Design*" (Concord, MA), e su "*Radio Electronics*" (New Jork City, NY). I nostri libri vengono ora tradotti in italiano, tedesco, cinese, francese, giapponese e tailandese. Se vi interessa imparare gli elementi fondamentali dell'elettronica digitale, il Bugbook I e II vi illustreranno l'argomento e vi metteranno in grado di fare gli esperimenti ideati per gli studenti. Il Bugbook IIA è stato scritto specificatamente per gli ingegneri, gli scienziati e per coloro che, come hobby, sono interessati alle tecniche di comunicazione asincrona usando l'UART. Il Bugbook IIA comprende anche gli esperimenti che vi permetteranno di usare le tecniche di comunicazione asincrona per interfacciare determinate apparecchiature ai computer o ad altri dispositivi che si basino sull'uso dell'UART.
Sul microprocessore/microcomputer 8080 sono stati scritti tre libri, il Bugbook III e i Bugbook V e VI. Questi ultimi due Bugbook costituiscono un'unica pubblicazione divisa in due volumi. Questi testi per autodidatti descrivono le tecniche di interfacciamento hardware e software, ed il modo in cui vengono impiegate per la risoluzione di problemi; la sostituzione del software con l'hardware; e molti altri argomenti interessanti. Abbiamo anche ideato la nuva serie, "Bugbook Applications". Ad essa appartengono attualmente: "Il timer 555: funzionamento, applicazioni ed esperimenti", "La progettazione dei filtri attivi, con esperimenti", "La progettazione degli amplificatori operazionali, con esperimenti," ed ora "DBUG: un programma interprete per la messa a punto dell'8080".

I seminari hanno sempre rivestito un ruolo importante nelle attività didattiche della Tychon. Ai nostri seminari, hanno partecipato moltissimi gruppi, fra cui Analog Devices, Inc., Xerox Corporation, Defense and Readiness Command, Illinois Central College e il Pratt Institute. Sono stati tenuti dei seminari anche in Italia, Inghilterra, Olanda, Australia e Svizzera. Se siete interessati a partecipare ad uno dei nostri seminari, potete mettervi in contatto con noi. In Italia potete rivolgervi a "La Scuola di Elettronica", Via Vittor Pisani 22 - 20124 MILANO, Tel. (02)6573050, 6572815, con la quale siano in stretto e costante contatto: la d.ssa Marina Baroni vi darà tutte le informazioni di vostro interesse. La Tychon stampa

anche la 8080 Octal Code Card e la 8080 Xex Code Card (distribuite in Italia dalla Microlem Divisione Didattica) che sono di notevole aiuto per la programmazione e la messa a punto dei programmi con l'8080. Contengono tutti i codici mnemonici dell'8080 ed i codici operativi ottali ed esadecimali ad essi corrispondenti. Tutti i vari gruppi di istruzioni sono associati con dei colori, per indicare, nel caso vi siano dei flags, quali di essi vengono coinvolti quando viene eseguita un'istruzione. Queste cartoline tascabili hanno anche una tabella per tutti i 128 caratteri, del codice ASCII la configurazine della parola di stato e delle coppie di registri dell'8080.

Cristopher A. Titus e Jonathan A. Titus

Sommario

DBUG: Un programma interprete per la messa a punto del software 8080

UN'INTRODUZIONE ALLA PROGRAMMAZIONE

Con una produzione di microcomputer che aumenta di giorno in giorno, molto utenti si rendono conto che sono necessarie due cose perché un microcomputer possa operare in modo corretto, l'hardware e il software. L'hardware è costituito dal microcomputer e da tutti i dispositivi elettronici periferici collegati al microcomputer stesso. Il microcomputer viene costruito intorno ad un circuito integrato microprocessore (8080 Z-80, 6800, 6502, F8 ecc.)ed alla memoria di lettura/scrittura, nonché di sola lettura, del microcomputer. I dispositivi periferici possono essere convertitori D/A e A/D, clock in tempo reale, dischi flessibili, CRT, telescriventi, o lettori/perforatori di banda ad alta velocità. Se vi occorre un microcomputer, potete magari trovare alcuni costruttori che possono fornirvi tutti i pezzi che vi saranno necessari per assemblare un microcomputer. Il software del microcomputer, ovvero i programmi, sono tutt'altra cosa. Dato che le vostre applicazioni per il microcomputer sono probabilmente uniche, non troverete a vostra disposizione un software già costituito che possa risolvere i vostri problemi. Perciò, dovrete sviluppare la vostra creatività e scrivere da soli i vostri programmi, o software. Questo problema può essere affrontato a livelli diversi di sofisticazione.

Potete programmare il microcomputer in *linguaggio macchina. Tale procedura consiste nel memorizzare gli 1 e gli 0 logici nella memoria del microcomputer.* Questo metodo richiede di solito pochissimi dispositivi periferici addizionali, o addirittura nessuno. Non avrete bisogno di un floppy disk o di una stampante ad alta velocità. Vi sono comunque degli inconvenienti nella programmazione in linguaggio macchina. È molto difficile ricordare i numeri binari. Preferireste dire "Vorrei tre dozzine di arance" oppure "Vorrei 100100_2 arance"? Se avete un programma di mille passi in linguaggio macchina, dovrete immagazzinare nella memoria del microcomputer mille numeri binari. Non solo è facile commettere degli errori inserendo tutti questi numeri, ma, se vi sono degli errori nel vostro programma, sarà molto difficile scoprirli. Potete programmare in linguaggio macchina anche usando numeri ottali, decimali o esodecimali. In questo modo, sarà più facile ricordare i codici delle verie istruzioni del microcomputer, ma la programmazione risulta ancora difficile. L'unico vantaggio della programmazione in linguaggio

macchina, è che richiede pochissima memoria. Se il vostro programma ha una lunghezza di duecento istruzioni, vi occorreranno soltanto duecento locazioni di meoria. Inoltre, per quanto riguarda i programmi brevi, risulta più veloce generare a mano i pogrammi in linguaggio macchina, anziché usare un editor o un assembler simbolici.

Un gradino più su della programmazione in linguaggio macchina, sta la programmazione in *linguaggio assembler.* La programmazione in linguaggio assembler viene realizzata usando i *codici mnemonici.* I *codici mnemonici* sono semplici simboli costituiti da tre a sette lettere, quali LXIH, STA, IN e MOVAB. Tali codici mnemonici sono più facili da ricordare per il programmatore e sono direttamente collegati alle operazioni che possono essere eseguite dal microcomputer. Un insieme, o un elenco, di codici mnemonici, costituisce un *programma sorgente.* Quando scrivete un programma su di un pezzo di carta, creerete probabilmente un programma sorgente usando codici mnemonici. Fungerete anche da *"editor"* perché aggiungerete, cancellerete e cambierete i codici mnemonici finché il programma non risulterà corretto. Man mano che aumenterà la vostra abilità nel programmare, probabilmente vi stancherete di variare il vostro programma sulla carta. Potrete invece usare un *programma editor simbolico,* che può essere eseguito sul vostro microcomputer 8080. Tale programma vi permette di manipolare i gruppi di codici mnemonici immagazzinati in quel momento nella memoria del microcomputer, nello stesso modo in cui trasferite, cambiate, inserite e cancellate i codici mnemonici sulla carta. Il microcomputer non può comunque *eseguire* un programma simbolico. I simboli devono invece essere convertiti nei loro equivalenti binari e memorizzati nel microcomputer prima di poter eseguire il programma.

Un 00100001 immagazzinato nella memoria del microcomputer equivale ad un LXIH per coloro che programmano in linguaggio assembler. Come ha un luogo la conversione del codice mnemonico LXIH in 00100001? Se il vostro programma fosse scritto su di un pezzo di carta, potreste andare a cercare l'istruzione LXIH in una tabella (Appendice C) e trovare l'equivalente binario. Potreste anche servirvi di un ausilio alla programmazione, come la 8080 HEX CODE CARD o la 8080 OCTAL CODE CARD della Tychon, distribuiti per l'Italia dalla divisione didattica della MICROLEM per determinare l'esatto codice binario (ottale o esadecimale) dell'istruzione LXIH. Questo processo può essere molto noioso ed anche facile ad errori. Così come c'erano degli *editor simbolici* per aiutarvi nella creazione di un programma, vi sono anche *programmi assemblatori* che possono eseguire il processo di conversione di codici simbolici in numeri binari, ottali o esadecimali. Il compito dell'assemblatore è quello di determinare l'esatto codice binario, ottale o esadecimale dell'istruzione LXIH e di tutte le altre istruzioni dell'8080. Quando l'assemblatore ha determinato l'esatto valore dell'istruzione LXIH, tale valore deve essere conservato da qualche parte. A seconda dell'assemblatore, il valore 00100001 verrà conservato su di un nastro, su una cassetta magnetica o su un floppy disk. In un secondo tempo, questa versione assemblata del programma, chiamata anche *programma oggetto o programma binario* (una serie di 1 o di 0), può essere memorizzata nel microcomputer ed eseguita. Fatto abbastanza sorprendente, il programma assemblatore assembla il programma sorgente nello stesso modo in cui lo assemblate voi quando è scritto su di un pezzo di carta. Quando assemblate il programma manualmente, cercate il codice mnemonico in una tabella, trovate

l'esatto valore binario, ottale o esadecimale, e lo scrivete sulla carta (un metodo di memorizzazione). Quando il programma viene assemblato dal microcomputer, quest'ultimo cerca una tabella interna all'assembler ed il programma sorgente. Se tale corrispondenza si verifica, il microcomputer preleva dalla memoria i byte binari, ottali o esadecimali appropriati contenuti nella tabella. Se non si verifica nessuna corrispondenza, l'assembler stampa di solito un messaggio di errore.

L'uso di programma assemblatore presenta due vantaggi fondamentali. Il primo è la velocità. Occore solo un millisecondo perché l'assemblatore determini l'esatto codice binario, ottale o esadecimale di un particolare codice mnemonico. Ad un programmatore che usi carta e penna, occorrono per lo meno due o tre secondi per portare a termine lo stesso processo. Il programma assemblatore può eseguire anche milioni e milioni di queste "traduzioni" senza fare un solo errore. E ciò è chiaramente meglio del metodo carta-e-penna.

Programmando sempre di più in linguaggio assembler vi accorgerete che vi saranno sempre più ripetizioni. Le subroutine della telescrivente saranno probabilmente le stesse di programma in programma, poiché si tratterà di sobroutine che effettuano il trasferimento di blocchi di dati o di sobroutine che effettuano ricerche fra un blocco di dati e l'altro. Sarebbe un'ottima cosa per voi avere un codice mnemonico o un comando che rappresentasse un programma di 100 o 200 passi in linguaggio assembler. Questo è esattamente ciò che i programmi BASIC, FORTRAN, e PL/1 vi permettono di fare.

Nel BASIC, potreste trovare un'istruzione di questo tipo: PRINT "THIS IS A TEST".

Quando il programma viene infine eseguito, la parola PRINT farà sì che venga eseguita una sobroutine in linguaggio assembler che stampa il messaggio THIS IS A TEST su di un CRT o una telescrivente. Il programmatore non deve preoccuparsi di come ciò avvenga o di quali registri interni all'8080 siano disponibili per essere usati.

In genere, la maggior parte dei programmi si possono suddividere in programmi *interpreti* e *compilatori* (interpreter e compiler). Un programma di tipo interpreter interpreta una linea di un programma alla volta. La maggior parte dei programmi BASIC che si possono usare con l'8080, sono programmi interpreter. Un *compiler* compila o traduce l'intero programma e produce una forma oggetto, o binaria, del programma stesso. Un compilatore FORTRAN compila le istruzioni FORTRAN e conserva la nuova versione del programma su nastro magnetico o su un blocco di schede. In un secondo tempo, questo blocco di schede o questo nastro magnetico potranno essere memorizzati nel computer ed eseguiti senza la presenza del compilatore FORTRAN. Per quanto riguarda i microcomputer 8080, quando viene eseguito un programma BASIC, l'interpreter BASIC deve contemporaneamente trovarsi in memoria.

Ciò che vorremmo farvi ora rilevare è che DBUG è un interpreter in linguaggio macchina. Ogni riga o numero di comandi viene interpretato così come voi lo avete scritto. Se DBUG fosse un compilatore, potreste scrivere in esso tutti i vostri comandi e poi, alla fine della giornata, eseguirne uno solo. Verrebbe quindi perforato un nuovo nastro, in base a tutti i dati e comandi che sono stati forniti. La maggior parte degli editor simbolici per i microcomputer utilizzanti l'8080 sono interpreter, perchè i comandi o le righe del testo vengono inseriti uno alla volta.

D'altra parte, gli assemblatori sono compilatori in quanto l'assemblatore, produce un nuovo programma assemblato oggetto o binario. Infine, BASIC e FORTRAN possono essere sia interpreter che compiler. Comunque, i programmi BASIC sono in genere interpreter e i programmi FORTRAN sono in genere compiler. Ricordate che questi sono i programmi che eseguirete sul vostro microcomputer. Non si tratta dei programmi sorgente in FORTRAN o in BASIC.

DBUG: UN'INTRODUZIONE

Il programma DBUG è un potente strumento di programmazione e di messa a punto dei programmi, sia per i neofiti che per i programmatori esperti. Per mezzo della tastiera della telescrivente o del lettore di banda, si possono facilmente inserire nel microcomputer brevi programmi. È possibile esaminare il contenuto di tutte le 65.536 locazioni di memoria di un microcomputer 8080, e scrivere nuovi valori per conservarli nella memoria di lettura/scrittura (R/W).

Una volta che un programma o un numero di programmi e di subroutine sono stati inseriti in un microcomputer, è possibile specificare qualunque indirizzo d'inizio di un programma. Ciò significa non essere più limitati da RESET, che costringe ad iniziare l'esecuzione del programma all'indirizzo di memoria 000 000. È possibile anche porre in un programma, che risiede nella memoria di lettura /scrittura, un breakpoint, che permette di eseguire il programma alla massima velocità, fino ad arrivare, appunto, al breakpoint. Quando il breakpoint è raggiunto, vengono visualizzati il contenuto dei registri A,B,C,D,E,H e L, il registro dei flag, la locazione di memoria indirizzata dai registri H e L, lo stak pointer e gli ultimi due valori carichi nello stack. Da questo punto in poi, potete proseguire nell'esecuzione del programma alla massima velocità, *procedere con la tecnica del passo passo*, o ritornare indietro ai passi relativi all'alterazione del programma, di cui abbiamo parlato in precedenza. Usando la tecnica del passo passo, potete eseguire una sola istruzione completa alla volta, che sia ad uno, a due o a tre byte, notando l'effetto che tale istruzione ha su tutti i registri, sulla locazione di memoria indirizzata dalla coppia di registri H ed L, sullo stack pointer, e sugli ultimi due valori caricati nello stack. Dopo aver proseguito nel programma con la tecnica del passo passo per un certo numero di istruzioni, potete continuare l'esecuzione passando di nuovo alla massima velocità o alterare il programma memorizzato nella memoria di lettura/scrittura. (1)

Potete anche perforare su nastro un programma contenuto in qualunque area della memoria. La ruotine di perforazione non è orientata a blocchi, per cui è possibile perforare tutto in una volta un file continuo di qualunque lunghezza. La banda perforata può essere memorizzata in qualunque momento nel microcomputer usando DBUG.

Avendo concluso questa breve introduzione a DBUG, prendiamo ora in considerazione i programmi DBUG, nonché alcuni esempi di programmazione.

---

(1) Nota: Non tutte le istruzioni possono essere eseguite con la tecnica del passo passo. Si tratta delle istruzioni di salto, di richiamo, di rientro e di ripristino. Comunque, ciò non sminuisce la potenza di DBUG, e noi vi mostreremo esempi di programmi che contengono queste istruzioni, ed il modo in cui possono essere eseguite.

I COMANDI DI DBUG

Quando viene avviato inizialmente, DBUG risponderà sempre con un punto di domanda,?, ed un ritorno carrello/spaziatura verticale o linee-feed (CRLF). Siete ora nel COMMAND MODE e solo i seguenti caratteri sono comandi legali:

P	Perfora un nastro. (Nota: devono essere specificate le informazioni dell'indirizzo).
R	Leggi un nastro
K	Elimina un breakpoint
S	Procedi passo passo. Esegui un'istruzione e visualizza il contenuto dei registri, ecc.
X	Procedi nell'esecuzione del programma
002 000 oppure 02 00	Accetta un indirizzo di memoria ottale o esadecimale (a seconda della versione di DBUG che state usando). Gli esempi qui presenti sono 002 000 (ottale) o 02 00 (esadecimale).

Se volete inserire in memoria un'istruzione o un byte di dati, dovete specificare un indirizzo a 16 bit nella forma di due byte ottali o esadecimali. Perciò, nel command mode, potete anche battere le informazioni dell'indirizzo. Per esempio, potete battere 002 000 o 02 00, dove il byte d'indirizzo HI è 002 (02 in codice esadecimale) e il byte d'indirizzo LO è 000 (00 in codice esadecimale). Il microcomputer visualizza automaticamente uno spazio dopo ogni parola ottale o esadecimale.

Per vedere che cosa conteneva la locazione di memoria 002 000, dovreste battere,

002 000 /

Il vostro ingresso (come utente di DBUG) viene sottolineato e sarà sottolineato in tutti gli esempi futuri. Ricordate che il microcomputer separa i byte ottali o esadecimali per mezzo degli spazi. In risposta alla barra slash,/, il microcomputer visualizza il contenuto della locazione di memoria da voi indirizzata:

002 000 / 125

Per cambiare il contenuto di questa locazione, dovreste digitare il byte ottale o esadecimale:

002 000 / 125 **231**

Potete ora battere un ritorno del carrello (CR), una spaziatura verticale (LF) o un altro numero. Supponete di aver fatto un errore di battitura e di volere in realtà *232* al posto di *231*. Tutto ciò che dovete fare è continuare a battere i valori ottali o esadecimali finché ottenete il valore esatto. Dovreste quindi avere, in codice ottale,

002 000 / 125 **231 241 232**

e in codice esadecimale,

02 00 / 55 **99 A1 9A**

Supponiamo ora di battere un CR o un LF. Un CR farà sì che il microcomputer metta in uscita un CRLF e che il programma rientri al COMMAND MODE in modo che possiate specificare un nuovo indirizzo o eseguire qualche altra operazione. Normalmente, quando rientrate al command mode, il microcomputer non visualizza un punto di domanda, ? . Verrà visualizzato un punto di domanda, ? , solo se cercate di digitare dei comandi illegali o dei digit ottali o esadecimali illegali. Ciò accadrebbe, ad esempio, se digitaste Q, 12P o AG.

Supponete di dover ora introdurre un programma di 11 passi nella memoria. Sapete che dovete specificare il primo indirizzo di memoria del programma, ma dovete sempre digitare un indirizzo oltre ai dati? *La rispota è no.*

Una volta che il microcomputer ha il primo indirizzo, lo incrementerà sempre e lo visualizzerà *a patto che venga digitata una spaziatura verticale (LF).* Perciò, per inserire il vostro programma di 11 passi, dovreste digitare quanto segue (vi mostriamo sia il listing ottale che quello esadecimale):

Esempio di Programma 1

Ottale	Mnemonico	Esadecimale
002 000 / 125 **076 LF**	MVIA	**02 00** / 55 **3E LF**
002 001 / 025 **001 LF**	001	02 01 / 15 **01 LF**
002 002 / 321 **006 LF**	MVIB	02 02 / D1 **06 LF**
002 003 / 056 **002 LF**	002	02 03 / 2E **02 LF**
002 004 / 077 **200 LF**	ADDB	02 04 / 3F **80 LF**
002 005 / 123 **200 LF**	ADDB	02 05 / 53 **80 LF**
002 006 / 276 **117 LF**	MOVCA	02 06 / BE **4F LF**
002 007 / 301 **127 LF**	MOVDA	02 07 / C1 **57 LF**
002 010 / 247 **025 LF**	DCRD	02 08 / A7 **15 LF**
002 011 / 105 **025 LF**	DCRD	02 09 / 45 **15 LF**
002 012 / 201 **166 CR**	HLT	02 0A / 81 **76 CR**

Notererete che DBUG rientrerà al command mode quando viene digitato il ritorno del carrello (CR) dopo l'istruzione 166 (HLT).

Allo scopo di determinare se il programma è stato digitato correttamente, potete specificare i byte d'indirizzo iniziali HI o LO e quindi digitare una barra. Dovrebbe così essere visualizzato il contenuto di quella locazione. Se digitate poi una spazia-

tura verticale (LF), saranno visualizzati l'indirizzo della locazione di memoria consecutiva, nonché il contenuto di memoria di quell'indirizzo. Si potrebbe continuare a digitare le spaziature verticali finché non viene elencato tutto il programma. Usando l'esempio precedente, che supporremo sia stato digitato e memorizzaato nella memoria di lettura/scrittura, potreste ottenere il listing seguente:

Ottale	Esadecimale
002 000 / 076 **LF**	**02 00** / 3E **LF**
002 001 / 001 **LF**	02 01 / 01 **LF**
002 002 / 006 **LF**	02 02 / 06 **LF**
002 003 / 002 **LF**	02 03 / 02 **LF**
002 004 / 200 **LF**	02 04 / 80 **LF**
002 005 / 200 **LF**	02 05 / 80 **LF**
006 006 / 117 **LF**	02 06 / 4F **LF**
002 007 / 127 **LF**	02 07 / 57 **LF**
002 010 / 025 **LF**	02 08 / 15 **LF**
002 011 / 025 **LF**	02 09 / 15 **LF**
002 012 / 166 **CR**	02 0A / 76 **CR**

Inutile a dirsi, potreste stancarvi di continuare a digitare le spaziature verticali, perciò abbiamo incorporato in DBUG una proprietà relativa al listing. Per usarla, dovete semplicemente digitare un indirizzo iniziale più la lettera L:

002 000 L o **02 00 L**

Il microcomputer inizierà allora ad elencare gli indirizzi di memoria ed i rispettivi contenuti iniziando dalla locazione di memoria da voi specificata. Dato che non avete specificato un indirizzo finale, come fa il microcomputer a sapere quando deve arrestarsi? *Per arrestare il listing, tutto quello che dovete fare è premere qualunque tasto della telescrivente,* tranne SHIFT e CRTL. Il microcomputer smetterà di visualizzare l'indirizzo e il contenuto attuale della locazione di memoria, e rientrerà quindi al command mode.

Avete imparato come inserire un programma per mezzo della telescrivente (TTY) o di un terminale ASCII adatto, come alterare il contenuto di una locazione di memoria e come elencare i programmi sia in codice ottale che esadecimale. DBUG è anche in grado di memorizzare e di estrarre i dati nel/dal nastro. Vengono usati a questo scopo i due comandi R e P, quando siete nel command mode.

COME SI LEGGE E SI PERFORA UN NASTRO

Per leggere un nastro, che deve essere nel format DBUG (Vedi appendice B), mettete il leader (parte iniziale, capo)del nastro (il leader consiste di una fila di perforazioni su un lato) nel lettore di banda della telescrivente, commutate l'interruttore del selettore su RUN, e digitate una R sulla telescrivente. Quando il nastro è stato letto completamente ed i valori dei dati sono stati memorizzati, DBUG rientra al command mode.

Per perforare un nastro, dovete conoscere gli indirizzi iniziale e finale dell'area di

memoria da perforare. Tali informazioni vengono inserite come quattro byte ottali o esadecimali, dove i primi due byte rappresentano l'indirizzo iniziale e gli ultimi due byte rappresentano l'indirizzo finale dei dati che devono essere perforati. Il contenuto della memoria viene quindi perforato su nastro, compreso il contenuto delle due locazioni d'indirizzo specificate, come segue:

P		**P**
002 000	(Indirizzo iniziale)	**02 00**
002 013	(Indirizzo finale)	**02 0B**

A destra viene usato il codice esadecimale, ed a sinistra viene usato il codice ottale. Prima o dopo che i byte di memoria appropriati vengono perforati, si perforano il capo (leady) e la coda (trailer) del nastro, come byte ottale 200 o come byte esadecimale 80. Il caricatore usato con il comando R in DBUG ignora il capo e la coda quando viene letto il nastro in questione.

COME SI ESEGUE UN PROGRAMMA

Una volta che il programma è stato caricato nella memoria di lettera/scrittura, eseguirlo è semplice. Dato che è possibile avere in memoria molti programmi e molte subroutine contemporaneamente, deve esserci un modo per specificare l'indirizzo iniziale di in dato programma. A questo scopo, dovete specificare un indirizzo a due byte, come avete fatto per perforare un nastro o inserire i dati in una locazione di memoria, e quindi digitare G (che sta per GO). Per dare inizio ad un programma all'indirizzo di memoria 002 000, dovreste digitare

002 000 G o **02 00 G**

Non appena digitate G, i registri A, B, C, D, E, H, e L vengono settati a 000 da DBUG ed il microcomputer inizia quindi ad eseguire il programma all'indirizzo 002 000.

Supponete che in memoria fosse immagazzinato il programma dell'esempio 2:

Esempio di Programma 2

003 000 / 041	LXIH	03 00 / 21
003 001 / 303	303	03 01 / C3
003 002 / 001	001	03 02 / 01
003 003 / 174	MOVAH	03 03 / 7C
003 004 / 323	OUT	03 04 / D3
003 005 / 000	000	03 05 / 00
003 006 / 175	MOVAL	03 06 / 7D
003 007 / 323	OUT	03 07 / D3
003 010 / 002	002	03 08 / 02
003 011 / 043	INXH	03 09 / 23
003 012 / 303	JMP	03 0A / C3
003 013 / 003	003	03 0B / 03
003 014 / 003	003	03 0C / 03

Dovreste iniziare l'esecuzione del programma digitando,

003 000 G o **03 00 G**

Quando il programma è in funzione, il contenuto dei registri H e L verrà messo in uscita sulle porte di uscita 000 e 002. Nel caso aveste dimenticato l'indirizzo iniziale del programma e l'aveste digitato nel modo sbagliato,

003 001 G o **03 01 G**

Fareste naturalmente in modo che DBUG esegua l'istruzione inziando dall'indirizzo di memoria 003 001. Comunque, il programma DBUG non sa che questo è un indirizzo iniziale sbagliato e che il contenuto di memoria a questo indirizzo 303 (C3) è il byte dei dati a otto bit meno significativo dell'istruzione LXIH. *Il programma DBUG interpreta questo byte di dati come se fosse un'istruzione!* Dato che un codice ottale 303 è il codice operativo di un'istruzione di salto incondizionato, i due byte successivi a 303 vengono interpretati in modo sbagliato da DBUG, come se rappresentassero i byte d'indirizzo LO e HI dell'istruzione di salto. L'8080 interpreta il contenuto della locazione di memoria 003 002 come se fosse il byte d'indirizzo LO, ed il contenuto della locazione di memoria 003 003 come se fosse il byte d'indirizzo HI. Perciò, l'8080 esegue un salto incondizionato alla locazione di meoria 174 001. Ovviamente, questo, non è ciò che volevate dall'8080. Perciò, è importante che ricordiate l'esatto indirizzo iniziale di tutti i vostri programmi. Se perforate un programma su nastro, troverete utile scrivere l'indirizzo iniziale direttamente sul nastro stesso.

L'USO DEL BREAKPOINT

Il *'breakpoint'* è una delle caratteristiche più utili di DBUG. Nella messa a punto di un programma, spesso si desidera che il programma sia eseguito normalmente fino ad una locazione di memoria predeterminata. Inoltre, può darsi che vogliate esaminare i registri interni all'8080 ed esaminare, e possibilmente modificare, il contenuto delle varie locazioni del vostro programma in cui sono memorizzati dati e istruzioni, a seconda dello stato dei registri interni dell'8080 o del contenuto di una particolare locazione di memoria. Per effettuare i passi relativi alle modifiche, DBUG si comporta nei riguardi del vostro programma come un *monitor di sistema.*

Voi potete decidere l'indirizzo al quale arrestere o "rompere" (break) il normale flusso del programma, e DBUG inserirà un'istruzione a questo indirizzo di breakpoint. Quando nel programma viene eseguita questa istruzione, il controllo viene automaticamente ritrasferito a DBUG. Quando ciò accade, DBUG conserva immediatamente tutti i valori del vostro programma nei registri interni all'8080, memorizzandoli nel proprio stack. Viene quindi visualizzato l'indirizzo a cui si è verificato il breakpoint, nonché il contenuto del byte dei flag, di tutti i registri, il contenuto di memoria indirizzato dalla coppia di registri H, lo stack pointer e gli ultimi due valori caricati nello stack dell'utente. A questo punto, potete trasferire i breakpoint e richiedere quindi a DBUG di proseguire nell'esecuzione del vostro programma.
DBUG ripristinerà quindi tutti i registri interni dell'8080 con i valori che essi avevano quando è stata eseguita l'istruzione di breakpoint, e proseguirà quindi nell'esecuzione del vostro programma finché non incontrerà di nuovo un breakpoint o finché il programma non verrà normalmente portato a termine. Potete

anche procedere passo passo attravero uno o più istruzioni individuali del vostro programma, usando il breakpoint, per vedere l'effetto che le varie istruzioni hanno sui registri interni all'8080.

Il breakpoint può essere posto in qualunque punto dei vostri programmi, *ma deve esser posto al primo byte delle istruzioni a più byte.* Inoltre il breakpoint non può essere posto sul primo byte di qualunque istruzione di salto o di richiamo, o di qualunque altra istruzione che non possa essere eseguita con la tecnica passo passo. Come porre un breakpoint nel vostro programma? Nell'esempio di programma 1, che abbiamo illustrato precedentemente nel paragrafo relativo ai comandi di DBUG, digitereste,

002 005 B o **02 05 B**

Questo verrebbe a porre il breakpoint (o il punto in cui interrompiamo la normale esecuzione del programma) sulla seconda istruzione ADDB. Se, dopo di questo, avete digitato,

002 000 G o **02 00 G**

Ecco come apparirebbe lo stampato sulla telescrivente (supponendo che l'uscita sia in codice ottale):

```
002 005 B
002 000 G
002 005
SZ 1 P 2   A   B   C   D   E   H   L   M      S P      C S
00000110  005 002 000 000 000 000 000 XXX  XXX XXX  XXX XXX
```

Notate che è stata eseguita l'istruzione all'indirizzo di breakpoint: il microcomputer ha sommato il contenuto del registro B (002) per due volte al contenuto del registro A (001), ottenendo come risultato 005. Il registro B contiene ancora 002 ed il resto dei registri è 000. I numeri 1 e 2 rappresentano i due bit del flag di carry nella parola dei flag. Il 2 rappresenta il riporto di un overflow del bit D7, il bit più significativo, e l'1 rappresenta lo stato del riporto relativo al quarto bit o ausiliario, fra i bit D3 e D4. Le "etichette" (label) dei registri non vengono stampate ogni volta che procedete attraverso il vostro programma con la tecnica del passo - passo, oppure ogni volta che settate un breakpoint, ma invece ogni cinque volte. Per il momento, ignorate il contenuto di SP e CS, di cui parleremo più avanti.

LA TECNICA DEL PASSO - PASSO

Digitando S (che sta per STEP), richiederete a DBUG di eseguire l'istruzione successiva, in questo caso un'istruzione MOVCA, e di stampare il registro, i flag, ecc. mostrando la conseguenza dell'esecuzione dell'istruzione. Notate che l'unico cambiamento si riscontra nel registro C, che prima conteneva 000 ed ora contiene 005, dopo l'istruzione di MOVCA. Se digitate di nuovo S, il microcomputer eseguirà l'istruzione successiva, MOVDA. L'unico cambiamento starà questa volta nel valore del registro D. Potete ora digitare una X per continuare l'esecuzione del

vostro programma. Dopo che è stata digitata la X, DBUG proseguirà nell'esecuzione del programma, finché non sarà stata eseguita l'istruzione HLT. Ora vi renderete probabilmente conto che non esiste alcun modo di procedere passo - passo attraverso un'istruzione HLT del vostro programma. Se siete in dubbio, potete provarci ugualmente usando il programma dell'esempio di programma 1. Per risparmiare tempo, iniziate il programma digitando,

002 010 B o **02 08 B**
002 000 G **02 00 G**

anziché

002 000 B o **02 00 B**
002 000 G **02 00 G**

Ciò vi porterà velocemente all'indirizzo 002 010 o 02 08 (esadecimale). Proseguite nell'esecuzione del programma con la tecnica del passo - passo.

Supponete che l'esempio di programma 3 sia già memorizzato nella memoria di lettura/scrittura.

Esempio di Programma 3

```
002 000 / 076    MVIA    02 00 / 3E
002 001 / 002    002     02 01 / 02
002 002 / 006    MVIB    02 02 / 06
002 003 / 003    003     02 03 / 03
002 004 / 007    RLC     02 04 / 07
002 005 / 200    ADDB    02 05 / 80
002 006 / 200    ADDB    02 06 / 80
002 007 / 303    JMP     02 07 / C3
002 010 / 004    004     02 08 / 04
002 011 / 002    002     02 09 / 02
```

Potete porre un brakpoint all'inizio del programma, effettuando il passo - passo, ma supponete di aver digitato:

002 015 B o **02 0D B**
002 000 G **02 00 G**

Potete digitare qualcosa sulla tastiera della vostra telescrivente? Il microcomputer sta ancora funzionando?

La risposta è che il microcomputer sta eseguendo il loop che voi avete nel vostro programma, ma il breakpoint non viene mai raggiunto, dato che esso è stato posto all'esterno del loop. *Comunque, il breakpoint c'è ancora!*

Abbiamo accennato in precedenza che potete usare il breakpoint di DBUG per mettere a punto i programmi memorizzati nella memoria di lettura/scrittura, ma non nella memoria di sola lettura. La ragione di questo sta nel fatto che DBUG

elimina dal vostro programma un'istruzione che è memorizzata nella memoria di lettura/scrittura e pone quindi la propria istruzione ad un solo byte nel vostro programma all'indirizzo di breakpoint. Dato che non si è raggiunto il breakpoint precedentemente posto, è semplice per voi determinare qual'è l'istruzione di breakpoint ad un solo byte, esaminando il contenuto della locazione 002 015 o 02 0D (esadecimale). A questo scopo, ripristinate DBUG e digitate quanto segue:

002 015 / o **02 0D /**

Dovreste osservare che l'istruzione di breakpoint usata da DBUG è 377 (esadecimale FF), che è l'istruzione RESTART 7 (RST 7) del set di istruzioni dell'8080.

Quando viene eseguita, l'istruzione RST 7 vettorizza il programma verso la locazione di memoria 000 070 (esadecimale 00 38). A questa locazione, c'è un'istruzione di salto che fa sì che il controllo di programma effettui un salto indietro al software di breakpoint in DBUG. Per questa ragione, quando usate DBUG, non dovreste mai eseguire un'istruzione di interrupt 377 (esadecimale FF).

Per poter capire il comportamento del comando K, fate quanto vi diciamo. Cambiate il contenuto della locazione 002 015 (esadecimale 02 0D) in qualunque valore che non sia 377 (esadecimale FF) eseguendo le operazioni seguenti:

002 015 / 377 **XXX CR** o **02 0D /** FF **XX CR**

DoveXXX (esadecimale XX) è il byte da voi scelto. Ora procederete in questo modo:

002 015 B		**02 0D B**
002 015 / CR	o	**02 0D / CR**

Che cosa viene stampato ora alla locazione di memoria 002 015? Naturalmente, l'istruzione di breakpoint 377. Ora digitate,

K		**K**
002 015 / XXX	o	**02 0D /** XX

Dovreste osservare il byte da voi scelto XXX (esadecimale XX). Ciò che avete fatto è stato porre un breakpoint alla locazione 002 015 (esadecimale 02 0D) e quindi, toccando il tasto K, eliminare il breakpoint ed inserire il vostro byte di dati originale.

Se il programma campione presentato in questo paragrafo non si trova ancora in memoria, digitatelo di nuovo e, in seguito, digitate quanto segue:

002 015 B	o	**02 0D B**
002 000 G		**02 00 G**

Dato che l'8080 non sta più eseguendo il programma DBUG, non potete usare il tasto K della vostra tastiera per eliminare il breakpoint. Comunque, ciò è possibile dopo aver ripristinato DBUG.

CHE COSA NON FARE CON DBUG

Abbiamo già accennato al fatto che non dovete cercare di procedere passo-passo attraverso un'istruzione di arresto (HLT). Esistono anche altri modi per fare sì che il vostro programma e/o DBUG vengano "buttati all'aria". La prima regola coinvolge le istruzioni a più byte e viene così formulata:
"Porre sempre il breakpoint al primo indirizzo di memoria di qualunque istruzione a più byte".

Considerate il programma dell'Esempio di Programma 4.

Esempio di Programma 4

002 000 / 076	MVIA	02 00 / 3E
002 001 / 250	250	02 01 / A8
002 002 / 041	LXIH	02 02 / 21
002 003 / 003	003	02 03 / 03
002 004 / 200	200	02 04 / 80
002 005 / 167	MOVMA	02 05 / 77
002 006 / 166	HLT	02 06 / 76

Non dovrete cercare di procedere con la tecnica del passo - passo attraverso questo programma digitando,

002 001 B o **02 01 B**
002 000 G **02 00 G**

Se tentate di farlo, il breakpoint non verrà mai raggiunto. Perché? Avete sostituito un'istruzione 377 (esadecimale FF) con l'istruzione che si trova all'indirizzo di memoria 002 001 (esadecimale 02 01), che rappresenta i *dati* per l'istruzione MVIA. Ciò che l'8080 ha fatto, eseguendo questo programma, è stato di memorizzare nel registro A il valore 377 (esadecimale FF) invece di 250 (esadecimale A8). Il programma è quindi proseguito finché non è stata eseguita l'istruzione HLT. Ciò che vogliamo farvi rilevare è che dovete prestare attenzione all'uso delle istruzioni aritmetico/logiche immediate, LXI, MVI, STA, LDA, LHLD, SHLD, ai salti e ai richiami, cioè a tutte le istruzioni a più byte. Potete procedere passo - passo attraverso tutte le istruzioni di salto, di richiamo, di rientro e di ripristino, ma DBUG sa se l'istruzione che è in esecuzione è un'istruzione a uno, due o tre byte.

Usando il programma campione mostrato più sopra, cambiate il contenuto di 002 001, riportandolo a 250, quindi procedete passo passo in modo corretto.

002 000 B o **02 00 B**
002 000 G **02 00 G**

Se digitate S, noterete che l'8080 esegue solo le istruzioni alle locazioni di memoria

002 000 o 02 00
002 002 02 02
002 005 02 05

SALTI, RICHIAMI, RESTART E RIENTRI

Basandovi sull'uso di un'istruzione HLT negli Esempi di Programma 1 e 4, riuscirete probabilmente a capire perché non potete procedere passo-passo attraverso qualunque istruzione di salto, richiamo, restart o rientro, usando DBUG. La ragione sta nel fatto che, se voi poteste, DBUG perderebbe il controllo dell'esecuzione del programma. Vediamo che cosa succede se fate questo tentativo. Useremo il programma campione dell'Esempio di Programma 5.

Esempio di Programma 5

```
002 000 / 061    LXISP    02 00 / 31
002 001 / 300    300      02 01 / C0
002 002 / 003    003      02 02 / 03
002 003 / 303    JMP      02 03 / C3
002 004 / 200    200      02 04 / 80
002 005 / 002    002      02 05 / 02
```

Iniziate l'esecuzione del programma usando,

```
                    002 000 B   o   02 00 B
                    002 000 G       02 00 G
002 000
SZ 1 P 2   A   B   C   D   E   H   L   M     S P     C S
01000110  000 000 000 000 000 000 000 303  003 300  001 100
S 002 003 ?
```

Osservate l'esecuzione dell'istruzione LXISP. L'istruzione successiva del vostro programma è un salto incondizionato. Tentate di procedere attraverso tale istruzione con la tecnica del passo - passo. Dovreste osservare un punto di domanda, ?, che è il modo usato da DBUG per dirvi che non è possibile procedere passo a passo attraverso un'istruzione di salto. Se sostituite uno qualunque dei salti (3X2, dove X = da 0 a 7), dei richiami (3X4, dove X = da 0 a 7), dei rientri (3X0, dove X = da 0 a 7), richiamo (315), rientro (311) o restart (3X7, dove X = da 0 a 7) con l'istruzione di salto, vedrete che DBUG si comporterà nello stesso modo. Notate che sono comprese tutte le istruzioni di restart, anche 377, perché si tratta dell'istruzione di restart che DBUG usa per settare il breakpoint.

Come è possibile mettere a punto programmi che contengono salti, richiami e rientri? Prendete in considerazione il programma seguente:

Esempio di Programma 6

```
002 000 / 021    LXID     02 00 / 11
002 001 / 012    012      02 01 / 0A
002 002 / 123    123      02 02 / 53
002 003 / 303    JMP      02 03 / C3
002 004 / 020    020      02 04 / 10
002 005 / 002    002      02 05 / 02

002 020 / 041    LXIH     02 10 / 21
002 021 / 252    252      02 11 / AA
002 022 / 125    125      02 12 / 55
002 023 / 166    HLT      02 13 / 76
```

Iniziate l'esecuzione del programma con,

```
                          002 000 B   o   02 00 B
                          002 000 G       02 00 G
002 000
01000110 000 000 000 123 012 000 000 303  003 170  000 106
```

Fate caso al contenuto di ogni registro. Dovreste osservare che solo i registri D e E sono non-zero. Ora digitate,

```
                          002 020 B   o   02 10 B
                          X               X
002 020
01000110 000 000 000 123 012 125 252 377  003 170  000 106

                          002 000 L   o   02 00 L
```

Digitando questi valori, il breakpoint viene trasferito all'indirizzo di memoria 002 020 (esadecimale 02 10) e a DBUG viene data l'istruzione di proseguire nell'esecuzione del programma dall'ultimo breakpoint, che si trovava all'indirizzo di memoria 002 000 (esadecimale 02 00). Notate che solo i registri H e L contengono dei valori nuovi, come conseguenza dell'esecuzione dell'istruzione LXIH all'indirizzo di memoria 002 020. *Avete eseguito l'istruzione di salto e DBUG controlla ancora l'esecuzione del programma!* Fate il listing del programma partendo da 002 000,

finché non superate l'istruzione HLT. Dov'è l'istruzione 377? La risposta è la seguente: DBUG elimina automaticamente il brekpoint (l'istruzione RST 7) e pone nuovamente nel vostro programma l'istruzione originale ogni volta che viene eseguita l'istruzione di breakpoint. La stessa tecnica che abbiamo usato per le istruzioni di salto può essere usata anche per le istruzioni di richiamo e di rientro. Il breakpoint viene trasferito nei passi software ai quali l'8080 trasferirà a sua volta il controllo.

LE OPERAZIONI RELATIVE ALLO STACK

Quando viene eseguita l'istruzione il breakpoint, DBUG usa temporaneamente due livelli (quattro byte di memoria) dello stack dell'utente. DBUG conserva quindi il vostro stack pointer e *stabilisce uno stack da usare per conto proprio.* Perciò, nei vostri programmi, potete fare qualunque cosa al vostro stack senza preoccuparvi che DBUG lo influenzi. Ricordate che, se nel vostro programma usate istruzioni di richiamo, di rientro, di inserimento o di estrazione, dovete caricare lo stack pointer con un'istruzione LXISP prima che la prima istruzione di richiamo, di caricamento o di prelievo venga eseguita. È buona norma eseguire un'istruzione LXISP come prima istruzione di *qualunque* programma. Comunque, se voi non costituite uno stack pointer, DBUG se ne costituirà uno per conto suo.

Quindi, i risultati che avete sinora ottenuto per quanto riguarda il contenuto di "SP" e di "CS", sono dovuti all'uso che DBUG fa dello stack. Non dovete supporre che questo stack possa essere usato per le vostre operazioni relative allo stack stesso. Se volete usare uno stack, dovete costruirvi il vostro stack per conto vostro.

Vediamo ora un esempio sulle operazioni relative allo stack. Inserite il programma seguente (a questo punto, non dovreste incontrare molte difficoltà):

Esempio di Programma 7

```
003 000 / 061    LXISP    03 00 / 31
003 001 / 100    100      03 01 / 40
003 002 / 003    003      03 02 / 03
003 003 / 041    LXIH     03 03 / 21
003 004 / 200    200      03 04 / 80
003 005 / 100    100      03 05 / 40
003 006 / 341    POPH     03 06 / E1
003 007 / 321    POPD     03 07 / D1
003 010 / 325    PUSHD    03 08 / D5
003 011 / 345    PUSHH    03 09 / E5
003 012 / 063    INXSP    03 0A / 33
003 013 / 063    INXSP    03 0B / 33
003 014 / 073    DCXSP    03 0C / 3B
003 015 / 041    LXIH     03 0D / 21
003 016 / 000    000      03 0E / 00
003 017 / 000    000      03 0F / 00
003 020 / 071    DADSP    03 10 / 39
003 021 / 000    NOP      03 11 / 00
003 022 / 000    NOP      03 12 / 00
003 023 / 000    NOP      03 13 / 00

003 100 / 001             03 40 / 01
003 101 / 002             03 41 / 02
003 102 / 003             03 42 / 03
003 103 / 004             03 43 / 04
003 104 / 005             03 44 / 05
003 105 / 006             03 45 / 06
003 106 / 007             03 46 / 07
```

Procedendo passo-passo attraverso questo programma, a partire dall'indirizzo di memoria 003 000, si possono vedere cambiamenti nello stack pointer e nel suo contenuto esattamente come ci aspettavamo dal programma:

```
003 000 B
003 000 G
003 000
SZ 1 P 2 A   B   C   D   E   H   L   M   S P     C S
01000110 000 000 000 000 000 000 000 303 003 100 002 001

S 003 003
01000110 000 000 000 000 000 100 200 115 003 100 002 001
```

```
S 003 006
01000110 000 000 000 000 000 002 001 015 003 102 004 003

S 003 007
01000110 000 000 000 004 003 002 001 015 003 104 006 005

S 003 010
01000110 000 000 000 004 003 002 001 015 003 102 004 003

S 003 011
SZ 1 P 2  A   B   C   D   E   H   L   M      S P     C S
01000110 000 000 000 004 003 002 001 015 003 100 002 001

S 003 012
01000110 000 000 000 004 003 002 001 015 003 101 003 002

S 003 013
01000110 000 000 000 004 003 002 001 015 003 102 004 003

S 003 014
01000110 000 000 000 004 003 002 001 015 003 101 003 000

S 003 015
01000110 000 000 000 004 003 000 000 303 003 101 003 000

S 003 020
SZ 1 P 2  A   B   C   D   E   H   L   M      S P     C S
01000110 000 000 000 004 003 003 101 000 003 101 003 000

S 003 021
01000110 000 000 000 004 003 003 101 000 003 101 003 000

S 003 022
01000110 000 000 000 004 003 003 101 000 003 101 003 000

S 003 023
01000110 000 000 000 004 003 003 101 000 003 101 003 000
```

Se usate un'istruzione di richiamo, con la stessa tecnica sviluppata per procedere passo-passo attraverso un'istruzione di salto, dovreste essere in grado di osservare l'indirizzo di rientro memorizzato nello stack. Prendiamo in considerazione il seguente programma:

Esempio di Programma 8

```
003 000 / 061     LXISP     03 00 / 31
003 001 / 100     100       03 01 / 40
003 002 / 003     003       03 02 / 03
003 003 / 076     MVIA      03 03 / 3E
003 004 / 240     240       03 04 / A0
003 005 / 315     CALL      03 05 / CD
```

```
003 006 / 020          TEST        03 06 / 10
003 007 / 003          0           03 07 / 03
003 010 / 000          NOP         03 08 / 00
003 011 / 000          NOP         03 09 / 00

003 020 / 006    TEST, MVIB        03 10 / 06
003 021 / 001          001         03 11 / 01
003 022 / 200          ADDB        03 12 / 80
003 023 / 200          ADDB        03 13 / 80
003 024 / 117          MOVCA       03 14 / 4F
003 025 / 311          RET         03 15 / C9
```

Procedete passo - passo attraverso questo programma ed osservate sia lo stack pointer che i due byte superiori dello stack stesso.

```
003 000 B
003 000 G
003 000
SZ 1 P 2  A   B   C   D   E   H   L   M    S P    C S
01000110 000 000 000 000 000 000 000 303 003 100 157 374

S 003 003
01000110 240 000 000 000 000 000 000 303 003 100 157 374
003 020 B
X
003 020
01000110 240 001 000 000 000 000 000 303 003 076 003 010

S 003 022
10000010 241 001 000 000 000 000 000 303 003 076 003 010

S 003 023
10000010 242 001 000 000 000 000 000 303 003 076 003 010

S 003 024
SZ 1 P 2  A   B   C   D   E   H   L   M    S P    C S
10000010 242 001 242 000 000 000 000 303 003 076 003 010

003 010 B
X
003 010
10000010 242 001 242 000 000 000 000 303 003 100 157 374

S 003 011
10000010 242 001 242 000 000 000 000 303 003 100 157 374
```

Avete ora visto tutti i comandi e le operazioni che DBUG può eseguire. Potete anche incominciare a capire perchè occorra un programma così lungo per realizzare compiti che appaiono tanto semplici. Un'ultima caratteristica, non menzionata in precedenza, è costituita dal fatto che DBUG vi permetterà di procedere passo - passo attraverso le operazioni di ingresso/uscita (I/O), semplificando così la messa a punto dei circuiti di interfaccia.

COME DARE INIZIO AL PROGRAMMA DBUG NELLA MEMORIA DI LETTURA/SCRITTURA

Locazioni di memoria usate da DBUG

020 000 – 023 377 Programma DBUG (memoria lettura/scrittura o PROM)
000 070 – 000 072 RST7 salta indietro a DBUG
003 300 – 003 377 Stack e memorizzazione temporanea

o

10 00 – 13 FF Programma DBUG (memoria di lettura/scrittura o PROM)
00 38 – 00 3A RST7 salta indietro a DBUG
03 C0 – 03 FF Stack e memorizzazione temporanea

Il bootstrap (lancio iniziale)

Per il caricamento da nastro di DBUG nella memoria di lettura/scrittura, è necessario avere in memoria un programma di lancio iniziale. I due programmi che seguono sono i listing di questo programma. Tali programmi si equivalgono dal punto di vista funzionale, la sola differenza sta nel fatto che un listing è in codice ottale e l'altro è in codice esadecimale. caricate in memoria uno o l'altro dei programmi in oggetto:

```
TYCHON EDITOR-ASSEMBLER V-2

        /CARICATORE DI LANCIO INIZIALE DELLA TYCON CON CHECKSUM
        /DI CONTROLLO V3, 21APR77, TYCHON, INC., BLACKSBURG, VA.
        /SE TALE LANCIO INIZIALE DEVE ESSERE TRASFERITO IN UN'ALTRA
        /PARTE DELLA MEMORIA ASSICURATEVI CHE L'INDIRIZZO DELLO STACK
        /POINTER RISPECCHI IL PUNTO IN CUI AVETE LA MEMORIA DI LETTURA
        /SCRITTURA. INOLTRE, DOVRETE CAMBIARE IL BYTE D'INDIRIZZO HI
        /DI TUTTE LE ISTRUZIONI DI SALTO E DI RICHIAMO. SUPPONENDO CHE
        /MANTENIATE GLI INDIRIZZI LO, DOVETE CAMBIARE GLI INDIRIZZI HI
        /AGLI INDIRIZZI LO DI 007, 014, 021, 024, 030, 034, 037, 044, 047, 057,
        /064, 067, 072, 075, 105, 113, e 126. PUO' DARSI CHE DOBBIATE CAMBIARE
        /LE ISTRUZIONI DI I/O PER RISPECCHIARE GLI INDIRIZZI DISPO-
        /SITIVI USATI CON IL VOSTRO SISTEMA.

                          *000 000
000 000 061  BOOT,        LXISP     /SETTA LO SP
000 001 350               350
000 002 000               000
000 003 026               MVID      /INIZIALIZZA IL CHECKSUM
000 004 000               000
000 005 315  LDRIN,       CALL      /LEGGI UN CARATTERE DAL NASTRO
000 006 116               READ
000 007 000               0
000 010 376               CPI       /È IL LEADER (200'S)?
000 011 200               200
000 012 312               JZ        /SI', CONTINUA A LEGGERE FINCHÈ
000 013 005               LDRIN     /NON LO OLTREPASSIAMO
000 014 000               0
000 015 376  CHKFRM,      CPI       /NON LO È: È IL FLAG DELL'INDIRIZZO?
000 016 100               100
```

```
000 017 302            JNZ        /NO, VEDI SE ERA IL FLAG DEL CHECKSUN
000 020 040            NOTA
000 021 000            0
000 022 315            CALL       /SI', ERA IL FLAG DELL'INDIRIZZO (A 100)
000 023 073            ADDCHK     /PRELEVA LE DUE RIGHE DI BANDA SEGUEN-
000 024 000            0          /TI, CHE SONO L'INDIRIZZO ALTO
000 025 147            MOVHA
000 026 315            CALL       /POI PRELEVA LE DUE RIGHE DI BANDA CHE
000 027 073            ADDCHK     /SONO L'INDIRIZZO LO
000 030 000            0
000 031 157            MOVLA
000 032 315  NXTIN,    CALL       /ORA PRELEVA UN'ALTRA RIGA DI BANDA
000 033 116            READ       /ED ELABORALA
000 034 000            0
000 035 303            JMP
000 036 015            CHKFRM
000 037 000            0
000 040 376  NOTA,     CPI        /NON ERA UN 100 PER UN INDIRIZZO
000 041 300            300        /È IL FLAG DEL CHECKSUM (300)?
000 042 302            JNZ
000 043 055            NOTSUM     /NO, ALLORA CONSIDERALA UN DATO
000 044 000            0
000 045 315            CALL       /ERA IL FLAG, PRELEVA LE DUE RIGHE DI
000 046 103            BYTE       /BANDA SEGUENTI, MA NON SOMMARE AD ESSE
000 047 000            0          /IL CHECKSUM
000 050 272            CMPD       /CONFRONTA IL CALCOLO FATTO CON IL VALORE
000 051 312  READOK,   JZ         /ATTUALE: SE SONO UGUALI, SALTIAMO
000 052 000            DBUG
000 053 020            0
000 054 166            HLT        / CHECKSUM NON ERANO UGUALI: ALT!
000 055 315  NOTSUM,   CALL       /
000 056 065            ADD1       /CONSIDERA LA RIGA DI BANDA ATTUALE E LA
000 057 000            0          /SEGUENTE COME DATI
000 060 167            MOVMA      /QUINDI CONSERVALE IN MEMORIA
000 061 043            INXH       /INCREMENTA IL POINTER DI MEMORIA E
000 062 303            JMP        /PRELEVA UN'ALTRA RIGA DI BANDA
000 063 032            NXTIN
000 064 000            0
000 065 315  ADD1,     CALL       /PRELEVA DUE RIGHE DI BANDA, 1 BYTE
000 066 106            BYTE1
000 067 000            0
000 070 303            JMP
000 071 076            ADDCHK+3
000 072 000            0
000 073 315  ADDCHK,   CALL
000 074 103            BYTE
000 075 000            0
000 076 365            PUSHPSW    /CONSERVA "A"
000 077 202            ADDD       /SOMMA LA SOMMA DI CONTROLLO
000 100 127            MOVDA      /E CONSERVALA DI NUOVO IN "D"
000 101 361            POPPSW     /PRELEVA DI NUOVO LA PAROLA DATI
000 102 311            RET
000 103 315  BYTE,     CALL       /LEGGI DAL NASTRO I DUE MSB
000 104 116            READ
000 105 000            0
000 106 017  BYTE1,    RRC        /FAI RUOTARE I BIT NEI MSB
```

```
000 107 017              RRC
000 110 117              MOVCA    /CONSERVA IL VALORE TEMPORANEO IN "C"
000 111 315              CALL
000 112 116              READ     /QUINDI LEGGI I 6 LSB
000 113 000              0
000 114 201              ADDC     /SOMMA I MSB
000 115 311              RET      /E RIENTRA CON IL VALORE IN "A"
000 116 323    READ,     OUT      /FORNISCI IMPULSI AL RELÈ DI CONTROLLO
000 117 021              021      /DEL LETTORE: SE LA TTY È COSI' EQUIPAG-
000 120 333              IN       /GIATA, PRELEVA LO STATO DEL RICEVITORE
000 121 021              021
000 122 346              ANI      /MASCHERA TUTTI I BIT ECCETTO I RICEVITORI
000 123 001              001
000 124 312              JZ
000 125 120              READ+2   /IL BIT È ANCORA UNO 0, CONTINUA AD ASPETTARE
000 126 000              0
000 127 333              IN       /I DATI SONO DISPONIBILI, INSERISCILI
000 130 020              020
000 131 311              RET      /RIENTRA CON LA LETTURA DEL CARATTERE IN "A"
```

TYCHON EDITOR-ASSEMBLER V-2

```
                  /CARICATORE DI LANCIO INIZIALE DELLA TYCON CON CHECKSUM
                  /DI CONTROLLO V3, 21APR77, TYCHON, INC., BLACKSBURG, VA.
                  /SE TALE LANCIO INIZIALE DEVE ESSERE TRASFERITO IN UN'ALTRA
                  /PARTE DELLA MEMORIA ASSICURATEVI CHE L'INDIRIZZO DELLO STACK
                  /POINTER RISPECCHI IL PUNTO IN CUI AVETE LA MEMORIA DI LETTURA
                  /SCRITTURA. INOLTRE, DOVETE CAMBIARE IL BYTE D'INDIRIZZO HI
                  /DI TUTTE LE ISTRUZIONI DI SALTO E DI RICHIAMO. SUPPONENDO CHE
                  /MANTENIATE GLI INDIRIZZI LO, DOVETE CAMBIARE GLI INDIRIZZI HI
                  /AGLI INDIRIZZI LO DI 07, 0C, 11, 14, 18, 1C, 1F, 24, 27, 2F, 34, 37, 3A
                  /3D, 45, 4B,56. PUO' DARSI CHE DOBBIATE CAMBIARE LE ISTRU-
                  /ZIONI DI I/0 PER RISPECCHIARE GLI INDIRIZZI DISPOSITIVI USATI
                  /CON IL VOSTRO SISTEMA.

                  *00H 00H
00 00 31  BOOT,   LXISP      /SETTA LO SP
00 01 E8          E8H
00 02 00          00H
00 03 16          MVID       /INIZIALIZZA IL CHECKSUM
00 04 00          00H
00 05 CD  LDRIN,  CALL       /LEGGI UN CARATTERE DAL NASTRO
00 06 4E          READ
00 07 00          0
00 08 FE          CPI        /È IL LEADER (80)?
00 09 80          80H
00 0A CA          JZ         /SI', CONTINUA AD ASPETTARE FINCHÈ OLTREPAS-
00 0B 05          LDRIN      /SIAMO TUTTO IL LEADER
00 0C 00          0
00 0D FE  CHKFRM, CPI        /NON LO È: È IL FLAG DELL'INDIRIZZO?
00 0E 40          40H
```

```
00 0F C2            JNZ        /NO, VEDI SE È IL FLAG DEL CHECKSUM (C0)
00 10 20            NOTA
00 11 00            0
00 12 CD            CALL       /SI', ERA IL FALG DELL'INDIRIZZO (A 40)
00 13 3B            ADDCHK     /PRELEVA LE DUE RIGHE DI BANDA SEGUENTI, CHE
00 14 00            0          /SONO L'INDIRIZZO ALTO
00 15 67            MOVHA
00 16 CD            CALL       /QUINDI PRELEVA LE DUE RIGHE DI BANDA CHE
00 17 3B            ADDCHK     /SONO L'INDIRIZZO LO
00 18 00            0
00 19 6F            MOVLA
00 1A CD  NXTIN,    CALL       /ORA PRELEVA UN'ALTRA RIGA DI BANDA
00 1B 4E            READ       /ED ELABORALA
00 1C 00            0
00 1D C3            JMP
00 1E 0D            CHKFRM
00 1F 00            0
00 20 FE  NOTA,     CPI        /NON ERA 40 PER UN INDIRIZZO
00 21 C0            C0H        /È IL FLAG DEL CHECKSUM (C0)?
00 22 C2            JNZ
00 23 2D            NOTSUM     /NO, QUINDI CONSIDERALO COME UN DATO
00 24 00            0
00 25 CD            CALL       /ERA IL FLAG, PRELEVA
00 26 43            BYTE       /LE DUE RIGHE DI BANDA SEGUENTI, MA NON
00 27 00            0          /SOMMARE AD ESSE IL CHECKSUM
00 28 BA            CMPD       /CONFRONTA IL CALCOLO FATTO CON I VALORI
00 29 CA  READOK,   JZ         /ATTUALI. SE SONO UGUALI, SALTIAMO
00 2A 00            DBUG
00 2B 10            0
00 2C 76            HLT        /I CHECKSUM NON ERANO UGUALI: ALT!
00 2D CD  NOTSUM,   CALL       /CONSIDERA LA RIGA DI BANDA ATTUALE E LA
00 2E 35            ADD1       /SUCCESSIVA COME DATI
00 2F 00            0          /QUINDI CONSERVALE IN MEMORIA
00 30 77            MOVMA      /INCREMENTA IL POINTER DELLA MEMORIA
00 31 23            INXH
00 32 C3            JMP        /E PRELEVA UN'ALTRA RIGA DI BANDA
00 33 1A            NXTIN
00 34 00            0
00 35 CD  ADD1,     CALL       /PRELEVA DUE RIGHE DI BANDA, 1 BYTE
00 36 46            BYTE1
00 37 00            0
00 38 C3            JMP
00 39 3E            ADDCHK+3
00 3A 00            0
00 3B CD  ADDCHK,   CALL
00 3C 43            BYTE
00 3D 00            0
00 3E F5            PUSHPSW    /CONSERVA "A"
00 3F 82            ADDD       /SOMMA IL CHECKSUM E
00 40 57            MOVDA      /CONSERVALA DI NUOVO IN "D"
00 41 F1            POPPSW     /PRELEVA DI NUOVO LA PAROLA DATI
00 42 C9            RET
00 43 CD  BYTE,     CALL       /LEGGI DAL NASTRO I DUE MSB
00 44 4E            READ
00 45 00            0
00 46 0F  BYTE1,    RRC        /FAI RUOTARE I BIT NEI MSB
```

```
00 47 0F              RRC
00 48 4F              MOVCA      /CONSERVA IL VALORE TEMPORANEO IN "C"
00 49 CD              CALL
00 4A 4E              READ       /QUINDI LEGGI I 6 LSB
00 4B 00              0
00 4C 81              ADDC       /SOMMA I MSB
00 4D C9              RET        /E RIENTRA CON IL VALORE IN "A"
00 4E D3   READ,      OUT        /FORNISCI IMPULSI AL RELÈ DI CONTROLLO DEL
00 4F 11              021        /LETTORE: SE LA TTY È COSI' EQUIPAGGIATA,
00 50 DB              IN         /PRELEVA LO STATO DEL RICEVITORE
00 51 11              021
00 52 E6              ANI        /MASCHERA TUTTI I BIT ECCETTO QUELLI DEI RICEVITORI
00 53 01              001
00 54 CA              JZ
00 55 50              READ+2     /IL BIT È ANCORA UNO 0, CONTINUA AD ASPETTARE
00 56 00              0
00 57 DB              IN         /I DATI SONO DISPONIBILI, INSERISCILI
00 58 10              020
00 59 C9              RET        /RIENTRA CON LA LETTURA DEL CARATTERE IN "A".
```

Dopo che è stato memorizzato il lancio iniziale, può darsi che dobbiate cambiare la subroutine del lettore di banda, che inizia all'indirizzo simbolico READ. Nella locazione di memoria 000 016 (esadecimale 00 4E) è memorizzata un'istruzione di uscita usata per fornire impulsi ad un relè di controllo del lettore di banda situato nella telescrivente. Se la vostra telescrivente non ha tale relè di controllo del lettore, dovreste porre due istruzioni NOP (il codice operativo è 000 o 00) nella locazione di memoria 000 016 e 000 017 (esadecimale 00 4E e 00 4F).

OTTALE			ESADECIMALE	
Indirizzo di memoria	Istru-zione	Codice mnemonico	Indirizzo di memoria	Istru-zione
000 116	323	OUT	00 4E	D3
000 117	021	<ADDR>	00 4F	11

L'istruzione d'ingresso memorizzata nella locazione di memoria 000 120 (esadecimale 00 50) viene usata per inserire il flag di dati disponibili dell'UART come parte della parola di stato a otto bit. Ciò indica, con un livello logico 1, che una parola è stata ricevuta dalla telescrivente. Può darsi che dobbiate cambiare il *codice indirizzo dispositivo* di questa istruzione, a seconda dell'indirizzo assegnato a questa porta d'ingresso del bit di stato nel vostro microcomputer. Tale indirizzo dispositivo è contenuto nella locazione di memoria 001 121 (esadecimale 00 51).

Può darsi che debba essere cambiato anche il byte di dati immediato dell'istruzione ANI che segue l'ingresso del bit di stato, a seconda di quale bit della porta d'ingresso di stato del vostro sistema viene assegnato al flag disponibile dei dati (DA) sull'UART. I byte di dati immediati che è possibile usare sono i seguenti:

Byte di Dati Immediato		Bit usato per il Flag disponibile dei dati	
Ottale	Esadecimale		
200	80	D7	(MSB)
100	40	D6	
040	20	D5	
020	10	D4	
010	08	D3	
004	04	D2	
002	02	D1	
001	01	D0	(LSB)

L'istruzione d'ingresso alla locazione di memoria 000 127 (esadecimale 00 57) inserisce la parola dati a otto bit proveniente dall'UART nel registro A dell'8080. L'indirizzo dispositivo contenuto nella locazione di memoria 000 130 (esadecimale 00 58) deve essere il codice indirizzo assegnato alla porta d'ingresso dei dati a otto bit dell'UART.

Se, per leggere il nastro di DBUG, non usate il lettore a nastro di una telescrivente, dovrete scrivere una nuova subroutine READ. A prescindere da come viene scritta la sobroutine, l'8080 dovrebbe uscire dalla subroutine stessa ,con il carattere a otto bit letto dal nastro, nel registro A. La subroutine deve usare solo i registri A e B. Dopo che sono state apportate tali modifiche, nel caso siano necessarie, ponete la parte iniziale della banda nel lettore di banda. La parte iniziale della banda è quella parte della banda stessa, detta anche nastro, in cui i fori vengono peforati uno dopo l'altro, e solo lungo un lato del nastro. La coda viene perforata nello stesso modo, ma naturalmente alla fine del nastro. Il capo e la coda del nastro (LDR, TRLR) vengono creati perforando sul nastro stesso il valore ottale 200 o il valore esadecimale 80. Dopo aver posto nel lettore di banda la parte iniziale della banda stessa, fissate il selettore del lettore nella posizione START, quindi premete e rilasciate l'interruttore RESET del vostro microcomputer. Se il nastro di DBUG è stato caricato correttamente, il caricatore del lancio iniziale darà automaticamente inizio al programma DBUG. DBUG stamperà quindi un punto di domanda, ?, sulla telescrivente e resterà in attesa dei vostri comandi. Se DBUG non viene caricato in modo corretto, il computer si arresterà. Se ciò accade, assicuratevi che il programma di lancio iniziale sia caricato correttamente e tentate quindi di caricare di nuovo DBUG in memoria.

COME DARE INIZIO AL PROGRAMMA DBUG IN PROM

Dopo aver aggiunto al vostro sistema 8080 le PROM di DBUG e dopo aver dato alimentazione, dovete porre nella memoria di lettura/scrittura del vostro microcomputer i tre byte istruzioni seguenti:

OTTALE			ESADECIMALE	
Indirizzo di memoria	Istruzione	Codice mnemonico	Indirizzo di memoria	Istruzione
000 000	303	JMP	00 00	C3
000 001	000	DBUG	00 01	00
000 002	020	0	00 02	10

Dopo aver inserito queste istruzioni, premete e rilasciate RESET. Tali istruzioni fanno sì che il microcomputer salti all'inizio di DBUG. DBUG, quando inizia, risponde con un punto di domanda, ?, sulla telescrivente e resta quindi in attesa dei vostri comandi.

COME CAMBIARE DBUG

Per comunicare con l'esterno, DBUG usa due porte d'ingresso e una porta di uscita. Di solito, tali porte di I/O sono collegate ad un UART o ad un USART, in modo che DBUG possa comunicare con un CRT o con una telescrivente. Una delle porte d'ingresso viene usata per inserire i dati ricevuti dall'UART (USART) e provenienti dal CRT o dalla telescrivente. L'altra porta d'ingresso viene usata per controllare lo stato dei flag di ricezione e di trasmissione dell'UART (o dell'USART). La porta di uscita viene usata dall'8080 per trasferire le informazioni all'UART (USART) in modo che esse vengano trasmesse e ricevute dal CRT o dalla telescrivente. Tutte queste funzioni si possono così riassumere:

Porta d'ingresso 020 (10)	Inserisce i dati ricevuti dall'UART (USART).
Porta d'ingresso 021 (11)	Inserisce i flag di stato dell'UART (USART). Questa viene spesso chiamata parola di stato.
Porta di uscita 020 (10)	Mette in uscita i dati sull'UART (USART).

Per indicare quando può essere trasmessa una parola proveniente dall'UART (USART), e quando una parola è stata ricevuta dall'UART (USART), si usano due bit della porta d'ingresso della parola di stato a otto bit. Per rilevare quando un carattere è stato ricevuto dall'UART (USART), si usa il bit meno significativo della porta d'ingresso di stato (il bit D0). Se tale bit è a livello logico 1, significa che è stato ricevuto un carattere. Quando il bit D2 di questa parola di stato è a livello logico 1, si può mettere in uscita sull'UART (USART) un altro carattere, in modo che esso venga automaticamente trasmesso al CRT o alla telescrivente. Vi mostriamo qui di eguito le convenzioni relative a tali bit dei flag:

Bit D0 = livello logico 1, l'UART (USART) ha ricevuto un carattere
Bit D2 = livello logico 1, l'UART (USART) può trasmettere un carattere

In DBUG, si usano le istruzioni ANI per determinare se uno dei bit di stato (D0 o D2) è o non è a livello logico 1 o 0.

Le locazioni di memoria seguenti contengono gli indirizzi della porta o di ingresso o di uscita, o i byte di dati immediati delle istruzioni ANI, usati per rilevare un bit di stato.

INDIRIZZI DI MEMORIA		CONTENUTO
Ottale	**Esadecimale**	
021 214	11 8C	Indirizzo della porta d'ingresso di stato
021 216	11 8E	Parola di stato per rilevare se è stato ricevuto un carattere
021 223	11 93	Indirizzo della porta d'ingresso usato per inserire il carattere ricevuto
021 275	11 BD	Indirizzo della porta di uscita degli impulsi del relé di conttrollo del lettore
021 320	11 D0	Indirizzo della porta d'ingresso di stato
021 322	11 D2	Parola di stato per rivelare se è stato ricevuto un carattere
021 327	11 D7	Indirizzo della porta d'ingresso usato per inserire il carattere ricevuto
021 340	11 E0	Indirizzo della porta d'ingresso di stato
021 342	11 E2	Parola di stato rivelare se è stato trasmesso il carattere
021 350	11 E8	Indirizzo della porta di uscita usata per trasmettere un carattere

Tutti gli utenti dovrebbero assicurarsi che tali locazioni di memoria contengano gli esatti indirizzi dispositivi e i codici dei bit di stato. Se il vostro sistema usa bit diversi all'interno della parola di stato, usate i byte di dati che ora vi mostriamo come byte di dati immediato dell'istruzione ANI usata per controllare i bit dei flag:

BYTE DI DATI IMMEDIATO		BIT USATI ALL'INTERNO DELLA PAROLA DI STATO
Ottale	**Esadecimale**	
200	80	D7 (MSB)
100	40	D6
040	20	D5
020	10	D4
010	08	D3
004	04	D2
002	02	D1
001	01	D0 (LSB)

Utenti del MITS/IMSAI 8080 (Microcomputer con bus S-100)

Se avete una memoria di lettura/scrittura dagli indirizzi di memoria 000 000 a 003 377 (esadecimale 00 00 - 03 FF) e da 020 000 a 023 377 (esadecimale 10 00 - 13 FF), non dovreste avere problemi nel caricamento e nell'uso di DBUG. Le uniche modifiche da apportare saranno nelle subroutine di ingresso/uscita. Tali modifiche sono descritte nel paragrafo precedente. Se il vostro microcomputer è equipaggiato con un USART, vi sono otto locazioni di memoria all'inizio di DBUG che potete usare per conservare le istruzioni di inizializzazione dell'USART. Vi raccomandiamo di usare un programma del tipo descritto nel paragrafo successivo.

Utenti dell'INTEL SBC 80/10, 80/20 e SDK 80

All'inizio di DBUG, abbiamo lasciato vuote otto locazioni di memoria, in modo da poter aggiungere la routine di inalizzazione di USART. Ecco come apparirà tale routine [1]:

---

(1) Per informazioni più dettagliate circa i bit e i flag di stato ecc., vi rimandiamo al Manuale Utenti del SBC-80/10, 80/20 o SDK-80 o ai dati della Programmable Communication Interface Intel 8251.

```
          MVIA
          MODE
          OUT
          CNCTL
          MVIA
          CMD
          OUT
          CNCTL
MODE = Istruzione di mode a otto bit
CNCTL = Indirizzo del registro di controllo dell'USART
CMD = Istruzione di comando a otto bit
```

Dovrete determinare gli esatti valori di MODE, CMD e CNCTL, a seconda del sistema 8080 che avete. Abbiamo aggiunto anche istruzioni NOP alla subroutine del lettore di banda della telescrivente. Per fornire impulsi al relè di controllo del lettore della telescrivente, potreste tentare:

```
READ,  MVIA
       TTYADV
       OUT
       CNCTL
       PUSHB
       LXIB
       <LO>
       <HI>
LOOP,  DCXB
       MOVAB
       ORAC
       JNZ
       LOOP
       0
       POPB
       MVIA
       CMD
       OUT
       CNCTL
         .
         .
         .
TTYADV = Fai avanzare il comando del nastro
CNCTL = Indirizzo del registro di controllo del USART
CMD = Arresta il comando di avanzamento del lettore
```

Dovrete nuovamente determinare i valori di TTYADV, CNCTL, e CMD a seconda del sistema 8080 che state usando. Dovrete inoltre determinare i valori dei due byte di dati dopo l'istruzione LXIB, usata per fornire un delay tramite il software. Può darsi che il vostro microcomputer 8080 sia più veloce o più lento rispetto al sistema di elaborazione usato da un altro utente, quindi può darsi che il vostro ritardo di tempo debba essere "regolato" con valori diversi. Le sole altre modifiche necessarie saranno quelle da apportare alle subroutine di ingresso/uscita. Tali modifiche sono state descritte nel primo paragrafo di CHANGING DBUG.

Utenti dell'MMD-1 della E&L Instruments

Le subroutine di I/O sono compatibili con gli indirizzi dispositivo assegnati agli UART sulla di scheda di Interfaccia di Memoria MMD-1/MI. Comunque, dovrete apportare quattro modifiche a DBUG perché avete una EPROM nelle prime 256 (512) locazioni di memoria. All'inizio di DBUG, c'è un indirizzo simbolico chiamato VECT, che sta ad indicare l'indirizzo di memoria 000 070 (esadecimale 00 38). Dato che tale indirizzo è posizionato nella EPROM KEX, dovrete usare le locazioni di memoria comprese fra 003 060 e 003 063 (esadecimale 03 30 - 03 33) per posizionare VECT, e non quelle comprese fra 000 070 e 000 073 (esadecimale 00 38 - 00 3B).

Cambiate il contenuto della locazione di memoria 020 014 (10 0C) da 070 in 060 (da 38 in 30). Cambiate anche il contenuto della locazione di memoria 020 015 (10 0D) da 000 in 003 (da 00 in 03). Dovete cambiare inoltre l'istruzione di restart che viene usata per il breakpoint. Cambiate il contenuto della locazione di memoria 020 314 (10 CC) da 377 in 367 (da FF in F7). Cambiate anche il contenuto della locazione di memoria 022 016 (12 0E) da 377 in 367 (da FF in F7).

La E&L Instruments Inc. mette a disposizione un set di quattro PROM 1702A (disponibili presso la Microlem Divisione Didattica - Milano) contenenti il programma di DBUG, che potrà essere impiegato insieme al vostro microcomputer MMD-1, che sia provvisto di una scheda in Interfaccia di Memoria MMD-1/MI.

L'uso di una diversa Istruzione di Restart per il Breakpoint

Se l'hardware da voi impiegato usa Restart 7 (RST 7), dovrete cambiare l'istruzione relativa al vettore DBUG (377 o esadecimale FF) in tre punti. Il primo cambiamento implica la modifica di uno dei byte di dati dell'istruzione LXID all'indirizzo di memoria 020 014 (esadecimale 10 0C). I valori dati dovrebbero essere cambiati in 010, 020, 030, 040, 050, o 060 (esadecimale 08, 10, 18, 20, 28, o 30) a seconda dell'istruzione di restart che intendete usare;

Istruzioni di restart da usare:	**Valori dati cambiati in:** (All'indirizzo 020 014 (10 0C))
RST 1 = 317 (CF)	010 (08)
RST 2 = 327 (D7)	020 (10)
RST 3 = 337 (DF)	030 (18)
RST 4 = 347 (E7)	040 (20)
RST 5 = 357 (EF)	050 (28)
RST 6 = 367 (F7)	060 (30)

Gli ultimi due cambiamenti coinvolgono l'istruzione attualmente usata per il breakpoint. Cambiate il contenuto delle locazioni di memoria 020 314 (10 CC) e 022 016 (12 OE) da 377 nel byte dell'istruzione di ripristino appropriata precedentemente mostrato.

ESEMPI

Abbiamo inserito tre esempi, in cui DBUG viene usato per procedere passo - passo attraverso un programma campione. Sarebbe impossibile per noi usare in questi esempi tutte le 244 istruzioni dell'8080. Noi abbiamo usato invece delle istruzioni che illustrano una particolare operazione o un particolare gruppo di operazioni che possono essere eseguite dall'8080

Se avete caricato DBUG nel vostro microcomputer 8080, potete servirvi di questi esempi per prendere confidenza con i comandi e le operazini di DBUG. Se non state usando DBUG, questi esempi vi daranno l'opportunità di valutare la potenza di DBUG stesso.

Il primo esempio illustra l'effetto che due delle operazioni di rotazione dell'8080 hanno sul flag di carry (riporto) e sul contenuto del registro A. Il secondo esempio usa le istruzioni di trasferimento dei dati per trasferire appunto i dati fra i registri interni all'8080 e la memoria di lettura/scrittura. L'ultimo esempio mostra come usare DBUG per procedere passo - passo attraverso un'istruzione di richiamo. In questo particolare esempio, è interessante osservare lo stack stesso.

Gli esempi suddetti possono essere così sintetizzati:

Esempio N. 1 Illustra l'effetto che due delle istruzioni di rotazione hanno sul flag di *riporto* e sul contenuto del registro A.
Esempio N. 2 Presenta alcune delle istruzioni di trasferimento dei dati.
Esempio N. 3 Mostra come procedere passo - passo attraverso un'istruzione di richiamo.

ESEMPIO N. 1

DIMOSTRAZIONE DI DUE DELLE ISTRUZIONI DI ROTAZIONE DELL'8080

Scopo

Questo esempio illustra l'effetto che due delle istruzioni di rotazione dell'8080 hanno sul contenuto del registro A e del flag di carry.

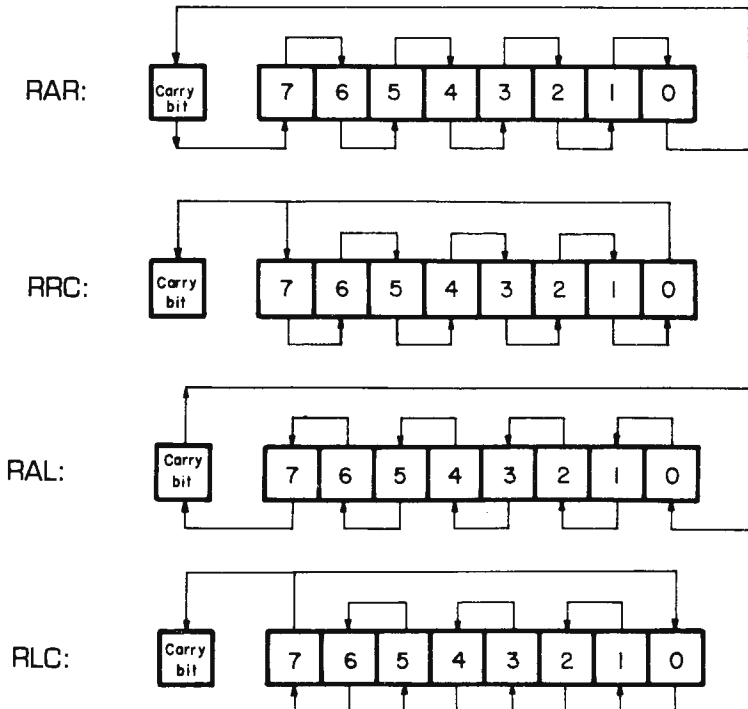
Programma

OTTALE			ESADECIMALE	
Indirizzo di memoria	Istruzione	Codice mnemonico	Indirizzo di memoria	Istruzione
003 000	076	MVIA	03 00	3E
003 001	004	<DATA>	03 01	04
003 002	017	RRC	03 02	0F
003 003	303	JMP	03 03	C3
003 004	002	<LO>	03 04	02
003 005	003	<HI>	03 05	03

Discussione

L'8080 è in grado di eseguire quattro diverse istruzioni di rotazione. Due di tali istruzioni fanno ruotare il contenuto del registro A verso destra e le altre due istruzioni fanno ruotare il contenuto del registro A verso sinistra. Il diagramma seguente sintetizza tali istruzioni e il modo in cui il flag di carry e il contenuto del registro A vengono influenzati da ognuna delle istruzioni suddette:

Basandoci sull'uscita della telescrivente, che abbiamo ottenuto quando ci siamo inoltrati passo - passo attravero il programma, è facile capire come l'istruzione RRC influenzi il registro A e il flag di carry. Così appariva il tabulato (osservate il contenuto del registro A e lo stato (un livello logico 1 o 0 del bit di carry 2):

```
003 000 B
003 000 G
003 000
SZ 1 P 2 A   B   C   D   E   H   L   M   S P   C S
01000110 004 000 000 000 000 000 000 303 004 000 200 000
```

```
S 003 002
01000110 002 000 000 000 000 000 000 303 004 000 200 000

003 002 B
X
003 002
01000110 001 000 000 000 000 000 000 303 004 000 200 000

003 002 B
X
003 002
01000111 200 000 000 000 000 000 000 303 004 000 200 000

003 002 B
X
003 002
01000110 100 000 000 000 000 000 000 303 004 000 200 000
```

L'istruzione MVIA nella locazione di memoria 003 000 (esadecimale 03 00) caricava il registro A con il valore dato 004 (esadecimale 04). L'istruzione RRC nella locazione di memoria 003 002 (esadecimale 03 02) faceva quindi ruotare il contenuto del registro A di un bit verso destra. Dopo l'esecuzione dell'istruzione RRC, il valore contenuto nel registro A era 002 (esadecimale 02). Dato che l'istruzione successiva a RRC era JMP, è stato ristabilito il breakpoint all'indirizzo di memoria 003 002 (esadecimale 03 02). Il breakpoint è stato inserito ripetutamente a quell'indirizzo per tutto il resto del testo.

Nel secondo passaggio attraverso il loop, il valore contenuto nel registro A veniva fatto ruotare da 002 a 001 (esadecimale da 02 a 01). Quando l'istruzione RRC è stata eseguita per la terza volta, il bit a livello logico 1 del registro A è stato fatto ruotare *sia* nel flag di carry (bit di carry 2) che nel bit più significativo (D7) del registro A. Questo è quanto ci si doveva aspettare, basandosi sul diagramma di flusso relativo all'istruzione RRC, che vi abbiamo precedentemente mostrato. Nel passaggio successivo attraverso il loop, il livello logico 1 del bit D7 veniva fatto ruotare nel bit D6, ed il riporto veniva azzerato a livello logico 0. Con il bit D6 a livello logico 1, il valore ottale del contenuto del registro A era 100 (esadecimale).

Cambiamo ora l'istruzione RRC in un'istruzione RLC, ottale 007 (esadecimele 07).

003 002 / 017 **007 CR** o **03 02** / 0F **07 CR**

Ci aspettiamo di vedere gli stessi risultati, ad eccezione del fatto che il livello logico 1 nel registro A dovrebbe essere fatto ruotare verso sinistra. Come potete vedere dal tabulato, La prima volta che l'istruzione RLC è stata eseguita, il contenuto del registro A è cambiato da 100 a 200 (esadecimale da 40 a 80). Con la seconda esecuzione dell'istruzione RLC, il bit in D7 è stato fatto ruotare verso sinistra *sia* nel flag di carry (bit di carry 2) che nel bit meno significativo del registro A, D0. L'ultima volta che sono stati eseguiti il loop e l'istruzione RLC, il contenuto del registro A è cambiato da 001 a 002 (esadecimale da 01 a 02).

```
003 002 B
X
003 002
SZ 1 P 2  A   B   C   D   E   H   L   M    S P    C S
01000110 200 000 000 000 000 000 000 303 004 000 200 000

003 002 B
X
003 002
01000111 001 000 000 000 000 000 000 303 004 000 200 000

003 002 B
X
003 002
01000110  002 000 000 000 000 000 000 303 004 000 200 000
```

ESEMPIO N. 2

DIMOSTRAZIONE DELLE ISTRUZIONI DI TRASFERIMENTO DEI DATI

Scopo

Questo esempio illustra il trasferimento dei dati fra i registri interni all'8080 e la memoria di lettura/scrittura.

Programma

OTTALE			ESADECIMALE	
Indirizzo di memoria	Istruzione	Codice mnemonico	Indirizzo di memoria	Istruzione
003 020	041	LXIH	03 10	21
003 021	200	<LO>	03 11	80
003 022	003	<HI>	03 12	03
003 023	066	MVIM	03 13	36
003 024	123	<DATA>	03 14	53
003 025	043	INXH	03 15	23
003 026	064	INRM	03 16	34
003 027	176	MOVAM	03 17	7E
003 030	053	DCXH	03 18	2B
003 031	167	MOVMA	03 19	77
003 032	116	MOVCM	03 1A	4E

Discussione

Abbiamo inserito il suddetto programma nella memoria di lettura/scrittura usando DBUG. Per osservare il flusso dei dati da e verso i registri dell'8080 e la memoria, abbiamo proseguito attraverso il programma con la tecnica del passo - passo.

```
003 020 B
003 020 G
003 020
SZ 1 P 2  A   B   C   D   E   H   L   M   S P   C S
01000110 000 000 000 000 000 003 200 001 004 000 200 000

S 003 023
01000110 000 000 000 000 000 003 200 123 004 000 200 000

S 003 025
01000110 000 000 000 000 000 003 201 001 004 000 200 000

S 003 026
00000010 000 000 000 000 000 003 201 002 004 000 200 000

S 003 027
00000010 002 000 000 000 000 003 201 002 004 000 200 000

S 003 030
SZ 1 P 2  A   B   C   D   E   H   L   M   S P   C S
00000010 002 000 000 000 000 003 200 123 004 000 200 000

S 003 031
00000010 002 000 000 000 000 003 200 002 004 000 200 000

S 003 032
00000010 002 000 002 000 000 003 200 002 004 000 200 000
```

Quando abbiamo posto il breakpoint all'indirizzo di memoria 003 002 (esadecimale 03 10)ed abbiamo dato inizio al programma a 003 020 (esadecimale 03 10) nella coppia di registri H erano memorizzati i valori dati 003 (esadecimale 03) del registro H e 200 (esadecimale 80) del registro L. L'istruzione successiva ha trasferito in modo immediato il valore dati 123 (esadecimale 53) nella locazione di memoria indirizzata dalla coppia di registri H. Notate il cambiamento a cui è sottoposto il contenuto della memoria (la colonna M del tabulato). L'istruzione INXH nella locazione di memoria 003 025 (esadecimale 03 15) ha quindi incrementato il contenuto del registro L (avrebbe incrementato anche il contenuto del registro H se il contenuto del registro L fosse stato incrementato da 377 a 000 (esadecimale da FF a 00), e una condizione di riporto sarebbe accorsa). Abbiamo osservato anche che il contenuto della memoria è cambiato quando l'indirizzo di memoria contenuto nella coppia di registri H è stato incrementato di 1.

Dopo l'esecuzione dell'istruzione INXH, è stata eseguita un'istruzione INRM. Tale istruzione ha fatto sì che il contenuto della locazione di memoria indirizzata dalla coppia di registri H venisse incrementato di 1. La locazione di memoria, indirizzata dalla coppia di registri H, quando questa istruzione è stata eseguita, era 003 201 (esadecimale 03 81). L'istruzione MOVAM ha quindi caricato nel registro A il contenuto di memoria indirizzato dalla coppia di registri H. Quindi l'istruzione DCXH ha fatto sì che il contenuto del registro L venisse decrementato di 1 (se il registro L fosse decrementato da 000 a 377 (esadecimale da 00 a FF), una condizione di riporto sarebbe accorsa e anche il contenuto del registro H verrebbe decre-

mentato di 1). Il valore dati contenuto nel registro A è stato quindi memorizzato nella locazione di memoria indirizzata dalla coppia di registri H. Perciò, il valore dati nel registro A è stato memorizzato nella locazione di memoria 003 200 (esadecimale 03 08). Infine, l'istruzione MOVCM ha fatto sì che il valore dati memorizzato nella locazione di memoria indirizzata dalla coppia di registri H, venisse caricato nel registro C.

ESEMPIO N. 3

USO DELLA TECNICA PASSO - PASSO CON UN'ISTRUZIONE DI RICHIAMO

Scopo

Questo esempio dimostra come sia possibile procedere passo - passo attraverso un'istruzione di richiamo usando la tecnica del passo - passo di DBUG.

Programma

OTTALE			ESADECIMALE	
Indirizzo di memoria	Istruzione	Codice mnemonico	Indirizzo di memoria	Istruzione
003 000	061	LXISP	03 00	31
003 001	100	<LO>	03 01	40
003 002	003	<HI>	03 02	03
003 003	076	MVIA	03 03	3E
003 004	240	<DATA>	03 04	A0
003 005	315	CALL	03 05	CD
003 006	020	<LO>	03 06	10
003 007	003	<HI>	03 07	03
003 010	000	NOP	03 08	00
003 011	000	NOP	03 09	00
003 020	006	MVIB	03 10	06
003 021	001	<DATA>	03 11	01
003 022	200	ADDB	03 12	80
003 023	200	ADDB	03 13	80
003 024	117	MOVCA	03 14	4F
003 025	311	RET	03 15	C9

Discussione

Procedendo passo - passo attraverso questo programma, abbiamo ottenuto il seguente tabulato:

```
003 000 B.
003 000 G
003 000
SZ 1 P 2  A   B   C   D   E   H   L   M    S P     C S
01000110 000 000 000 000 000 000 000 303 003 100 157 374

S 003 003
01000110 240 000 000 000 000 000 000 303 003 100 157 374

S 003 005 ?
003 005 / 315
003 006 / 020
003 007 / 003
003 020 B
X
003 020
01000110 240 001 000 000 000 000 000 303 003 076 003 010

S 003 022
10000010 241 001 000 000 000 000 000 303 003 076 003 010

S 003 023
10000010 242 001 000 000 000 000 000 303 003 076 003 010

S 003 024
SZ 1 P 2  A   B   C   D   E   H   L   M    S P     C S
10000010 242 001 242 000 000 000 000 303 003 076 003 010

S 003 025 ?
003 025 / 311
003 010 B
X
003 010
10000010 242 001 242 000 000 000 000 303 003 100 157 374

S 003 011
10000010 242 001 242 000 000 000 000 303 003 100 157 374
```

Notate che l'8080 ha seguito l'istruzione LXISP nella locazione di memoria 003 000 (esadecimale 03 00) e poi l'istruzione MVIA all'indirizzo 003 003 (esadecimale 03 03). Quando abbiamo cercato di procedere passo - passo attraverso l'istruzione CALL all'indirizzo 003 005 (esadecimale 03 05), DBUG ha stampato un punto di domanda, comunicandoci così che era stato fatto un tentativo di seguire una delle istruzioni passo - passo non eseguibili. Per scoprire di che istruzione si trattava, abbiamo esaminato la locazione di memoria 003 005 (esadecimale 03 05). In quella locazione di memoria era contenuta un'istruzione CALL (315, CD), quindi abbiamo esaminato anche le due locazioni di memoria consecutive successive per trovare l'indirizzo LO e HI della subroutine che veniva richiamata. L'indirizzo LO era 020 (esadecimale 10) memorizzato nella locazione di memoria 003 006 (esadecimale 03 06) e l'indirizzo HI era 003 (esadecimale 03) memorizzato nella locazione di memoria 003 007 (esadecimale 03 07). Dopo che il breakpoint è stato settato a

questo nuovo indirizzo, 003 020 (esadecimale 03 10), è stato usato il comando di continuazione (X) per proseguire nell'esecuzione del programma.

È stata quindi eseguita l'istruzione CALL, che ha portato il computer al breakpoint all'indirizzo di memoria 003 020 (esadecimale 03 10). Notate che, dopo aver eseguito CALL ed aver raggiunto il breakpoint, il registro B è stato caricato a 001 (esadecimale 01) per via dell'istruzione MVIB all'indirizzo di memoria 003 020 (esadecimale 02 10). Osserverete che anche lo stack pointer è decrementato di 2 a 003 076 (esadecimale 03 3E). Come dovremmo aspettarci, l'indirizzo di rientro (003 010, esadecimale 03 08) viene memorizzato nello stack, come mostra la colonna CS. Le due istruzioni successive sono entrambe ADDB. Abbiamo scoperto che il contenuto del registro A era 242 (esadecimale A2) quando siamo avanzati passo - passo ed abbiamo eseguito l'ultima istruzione ADDB nella locazione di memoria 003 023 (esadecimale 03 13). Quando abbiamo proceduto passo - passo attraverso l'istruzione conservata nella locazione di memoria 003 024 (esadecimale 03 14), il contenuto del registro A è stato trasferito nel registro C. Quando abbiamo cercato di procedere passo - passo attraverso l'istruzione all'indirizzo di memoria 003 025 (esadecimale 03 15), DBUG non ha eseguito l'istruzione. Quando abbiamo esaminato il contenuto della locazione di memoria 003 025 (esadecimale 03 25), abbiamo trovato un'istruzione di rientro incondizionato (RET = 311,C9).

Sapendo che l'istruzione RET estrae dallo stack l'indirizzo di rientro, abbiamo esaminato il contenuto dello stack stesso (CS) e abbiamo visto su di esso memorizzato, 003 010 (esadecimale 03 08), come indirizzo al quale settare il breakpoint successivo. È stato quindi usato il comando di prosecuzione (X) per procedere nell'esecuzione del programma, dopo che il breakpoint è stato settato all'indirizzo di memoria 003 010 (esadecimale 03 08). È stata quindi eseguita l'istruzione RET, e nel frattempo l'8080 ha prelevato l'indirizzo di rientro dallo stack, facendo sì che lo stack pointer venisse incrementato di 2.

Quando è stato incontrato il breakpoint all'indirizzo di memoria 003 010 (esadecimale 03 08), è stata eseguita l'istruzione a quell'indirizzo ed è stato stampato il contenuto dei registri. L'istruzione eseguita era NOP, ovvero nessuna operazione. Perciò, non è cambiato nessuno dei valori memorizzati nei registri. Quando abbiamo proseguito passo - passo attraverso l'istruzione all'indirizzo di memoria 003 011 (esadecimale 03 09), non è cambiato nessuno dei valori memorizzati in uno qualunque dei registri. Questo perché l'8080 ha eseguito un'altra istruzione NOP.

APPENDICE A

Subroutine generali a disposizione dell'utente

NOME	REGISTRO USATO	INDIRIZZO
BYTE	A,B,C	020 130; 10 58

Legge due righe dal lettore di banda della telescrivente. La prima riga di banda contiene i due bit più significativi e la seconda riga di banda contiene i sei bit meno significativi. Esce dalla sobroutine con il byte nel registro A. Per ulteriori informazioni, vedi l'appendice FORMAT DELLA BANDA PER DBUG.

NOME	REGISTRO USATO	INDIRIZZO
LDDR	A,B,C	020 235; 10 9D

Questa sobruotine perfora 10" di 200 ottale (esadecimale 80) sul perforatore di banda della telescrivente.

NOME	REGISTRO USATO	INDIRIZZO
BYTOUT	A,B,C	020 251; 10 A9

Perfora il contenuto del registro A sul perforatore di banda della telescrivente. I due bit più significativi vengano fatti ruotare nei due bit meno significativi del registro A e vengono quindi perforati su di una riga. I sei bit meno significativi rimanenti vengono poi perforati nella riga di banda consecutiva successiva.

NOME	REGISTRO USATO	INDIRIZZO
CRLF	A,B	021 075; 11 3D

Sulla telescrivente vengono stampati un ritorno del carrello e una spaziatura verticale (line - feed)

NOME	REGISTRO USATO	INDIRIZZO
OCTIN (HEXIN)	A,B,C,E	021 122; 11 52

Questa sobroutine viene usata per convertire numeri ottali (esadecimali) in numeri binari. Digitando un numero ottale a tre digit (esadecimale a due digit) nel registro E si troverà l'equivalente binario quando l'8080 rientra. *I numeri non validi faranno sì che il controllo di programma ritorni a DBUG.*

NOME	REGISTRO USATO	INDIRIZZO

HLOUT | A,B,C | 021 227; 11 97

Stampa il contenuto della coppia di registri H sotto forma di due numeri esadecimali o ottali separati da uno spazio.

OCTOUT (HEXOUT) | A,B,C | 021 234; 11 9C

Questa subroutine stampa il contenuto del registro A sotto forma di un numero esadecimale o ottale.

READ | A | 021 274; 11 BC

Questa subroutine fornisce impulsi al relè di controllo del lettore della telescrivente, inserisce da nastro il carattere a otto bit ed esce con esso nel registro A.

TTYI | A | 021 317; 11 CF

Inserisce il carattere proveniente dalla tastiera della telescrivente quando viene premuto un tasto. Esce con il carattere a otto bit nel registro A. Il carattere non viene stampato sulla telescrivente.

TTYIN | A,B | 021 331; 11 D9

Inserisce il carattere proveniente dalla tastiera della telescrivente quando viene premuto un tasto, stampa il carattere sulla telescrivente ed esce con il valore a sette bit nei registri A e B. Il bit più 021 336; 11 DE significativo verrà settato a 0.

TTYOUT | A,B | 021 336; 11 DE

Entra con il carattere da digitare nel registro A. Quando l'8080 rientrerà, il carattere sarà nei registri A e B.

SPC | A,B,C | 021 362; 11 F2

Stampa un solo spazio sulla telescrivente.

MORSPC | A,B,C | 021 364; 11 F4

Entra con il numero degli spazi (ottale o esadecimale) da stampare sulla telescrivente, nel registro C. L'8080 uscirà da questa sobroutine quando il contenuto del registro C sarà decrementato a 0.

BIT | A,B,C,E | 023 207; 13 87

Stampa il contenuto del registro E sotto forma di una stringa di uni e di zeri. Il bit più significativo sarà sulla sinistra, il bit meno significativo sarà sulla destra. Verrà battuto uno spazio dopo la stringa.

NOME	REGISTRO USATO	INDIRIZZO
NXTLET	A,B,C,H,L	023 267; 13 B7

Questa subroutine si usa per stampare i messaggi ASCII su una telescrivente. Inserite la subroutine con la coppia di registri H che punta il primo carattere della stringa di caratteri, che viene stampato. Per uscire da questa subroutine, è necessario memorizzare 000 (esadecimale 00) alla fine della stringa. Quando l'8080 rientrerà da questa subroutine, la coppia di registri H punterà questo 000 (esadecimale 00) alla fine della stringa. Una caratteristica particolare di questa subroutine è quella degli spazi multipli. In quel numero di spazi risulterà stampato qualunque carattere minore di 010 (esadecimale 08) che non sia 000.
I seguenti caratteri immagazzinati in memoria,

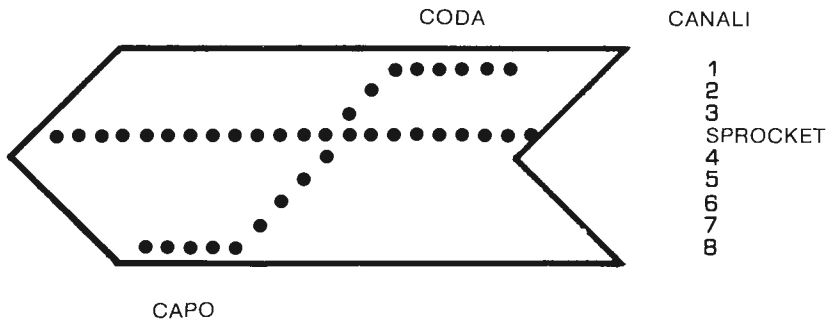
101, 102, 103, 003, 061, 062, 063, 215, 212, 002, 130, 131, 132, 000

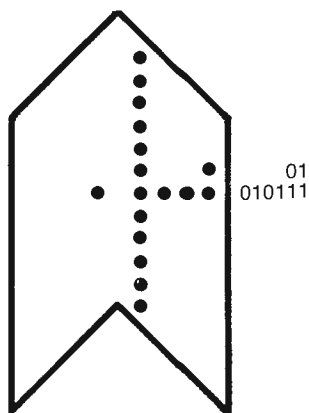
risulteranno così stampati:

```
ABC   123
  XYZ
```

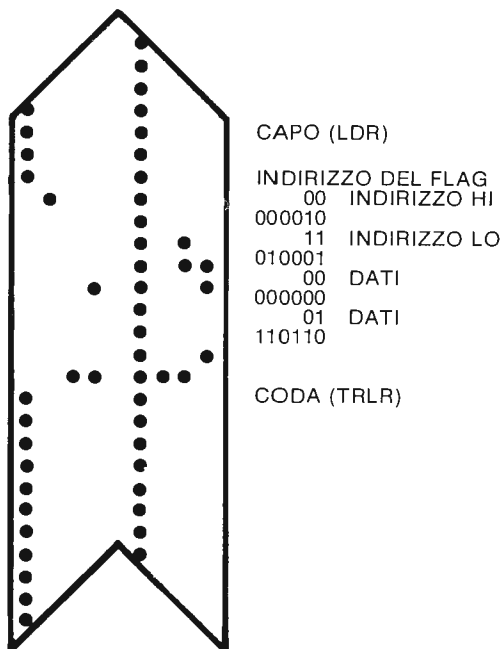
APPENDICE B

Format della banda per DBUG

Il format della banda è il seguente:

Il canale 8 (quando viene perforato da solo) rappresenta l'esca inziale/finale della banda (LDR, TRLR; ottale 200 o esadecimale 80). Quando il canale 7 viene perforato da solo, le quattro righe successive (parole) rappresentano le informazioni relative all'indirizzo. Le parole dati e le informazioni sugli indirizzi vengono suddivise in un byte a due bit e in un byte a sei bit. Perciò, 127 (esadecimale 57) apparirebbe come:

01 010 111 = 01 010111
0101 0111 = 01 010111

La banda seguente ha sia le informazioni dati che indirizzi. Un indirizzo di 002 321 (esadecimale 02 D1) sarebbe:

002 = 00 000 010 = 00 000010
321 = 11 010 001 = 11 010001

02 = 0000 0010 = 00 000010
D1 = 1101 0001 = 11 010001

APPENDICE C

Sommario del set di istruzioni 8080

Mnemonic	Description	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock[2] Cycles
		Instruction Code[1]								
MOV r1, r2	Move register to register	0	1	D	D	D	S	S	S	5
MOV M, r	Move register to memory	0	1	1	1	0	S	S	S	7
MOV r, M	Move memory to register	0	1	D	D	D	1	1	0	7
HLT	Halt	0	1	1	1	0	1	1	0	7
MVI r	Move immediate register	0	0	D	D	D	1	1	0	7
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10
INR r	Increment register	0	0	D	D	D	1	0	0	5
DCR r	Decrement register	0	0	D	D	D	1	0	1	5
INR M	Increment memory	0	0	1	1	0	1	0	0	10
DCR M	Decrement memory	0	0	1	1	0	1	0	1	10
ADD r	Add register to A	1	0	0	0	0	S	S	S	4
ADC r	Add register to A with carry	1	0	0	0	1	S	S	S	4
SUB r	Subtract register from A	1	0	0	1	0	S	S	S	4
SBB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4
ANA r	And register with A	1	0	1	0	0	S	S	S	4
XRA r	Exclusive Or register with A	1	0	1	0	1	S	S	S	4
ORA r	Or register with A	1	0	1	1	0	S	S	S	4
CMP r	Compare register with A	1	0	1	1	1	S	S	S	4
ADD M	Add memory to A	1	0	0	0	0	1	1	0	7
ADC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7
SUB M	Subtract memory from A	1	0	0	1	0	1	1	0	7
SBB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7
ANA M	And memory with A	1	0	1	0	0	1	1	0	7
XRA M	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7
ORA M	Or memory with A	1	0	1	1	0	1	1	0	7
CMP M	Compare memory with A	1	0	1	1	1	1	1	0	7
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7
ANI	And immediate with A	1	1	1	0	0	1	1	0	7
XRI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7
ORI	Or immediate with A	1	1	1	1	0	1	1	0	7
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7
RLC	Rotate A left	0	0	0	0	0	1	1	1	4
RRC	Rotate A right	0	0	0	0	1	1	1	1	4
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4
RAR	Rotate A right through carry	0	0	0	1	1	1	1	1	4
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10
JC	Jump on carry	1	1	0	1	1	0	1	0	10
JNC	Jump on no carry	1	1	0	1	0	0	1	0	10
JZ	Jump on zero	1	1	0	0	1	0	1	0	10
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	10
JP	Jump on positive	1	1	1	1	0	0	1	0	10
JM	Jump on minus	1	1	1	1	1	0	1	0	10
JPE	Jump on parity even	1	1	1	0	1	0	1	0	10
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	10
CALL	Call unconditional	1	1	0	0	1	1	0	1	17
CC	Call on carry	1	1	0	1	1	1	0	0	11/17
CNC	Call on no carry	1	1	0	1	0	1	0	0	11/17
CZ	Call on zero	1	1	0	0	1	1	0	0	11/17
CNZ	Call on no zero	1	1	0	0	0	1	0	0	11/17
CP	Call on positive	1	1	1	1	0	1	0	0	11/17
CM	Call on minus	1	1	1	1	1	1	0	0	11/17
CPE	Call on parity even	1	1	1	0	1	1	0	0	11/17
CPO	Call on parity odd	1	1	1	0	0	1	0	0	11/17
RET	Return	1	1	0	0	1	0	0	1	10
RC	Return on carry	1	1	0	1	1	0	0	0	5/11
RNC	Return on no carry	1	1	0	1	0	0	0	0	5/11
RZ	Return on zero	1	1	0	0	1	0	0	0	5/11
RNZ	Return on no zero	1	1	0	0	0	0	0	0	5/11
RP	Return on positive	1	1	1	1	0	0	0	0	5/11
RM	Return on minus	1	1	1	1	1	0	0	0	5/11

Mnemonic	Description	Instruction Code[1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock[2] Cycles
RPE	Return on parity even	1	1	1	0	1	0	0	0	5/11
RPO	Return on parity odd	1	1	1	0	0	0	0	0	5/11
RST	Restart	1	1	A	A	A	1	1	1	11
IN	Input	1	1	0	1	1	0	1	1	10
OUT	Output	1	1	0	1	0	0	1	1	10
LXI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
LXI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
LXI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
LXI SP	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	11
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	11
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	11
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	11
POP B	Pop register pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
STA	Store A direct	0	0	1	1	0	0	1	0	13
LDA	Load A direct	0	0	1	1	1	0	1	0	13
XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
XTHL	Exchange top of stack, H & L	1	1	1	0	0	0	1	1	18
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	5
PCHL	H & L to program counter	1	1	1	0	1	0	0	1	5
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	5
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	5
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	5
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	5
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	5
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	5
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	5
DCX SP	Decrement stack pointer	0	0	1	1	1	0	1	1	5
CMA	Complement A	0	0	1	0	1	1	1	1	4
STC	Set carry	0	0	1	1	0	1	1	1	4
CMC	Complement carry	0	0	1	1	1	1	1	1	4
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
EI	Enable Interrupts	1	1	1	1	1	0	1	1	4
DI	Disable interrupt	1	1	1	1	0	0	1	1	4
NOP	No operation	0	0	0	0	0	0	0	0	4

NOTES: 1. DDD or SSS – 000 B – 001 C – 010 D – 011 E – 100 H – 101 L – 110 Memory – 111 A.
2. Two possible cycle times, (5/11) indicate instruction cycles dependent on condition flags.

Per concessione della Intel Corporation,
Santa Clara, California 95051

APPENDICE D

Riassunto dei comandi

Quando è nel comand mode;

P — Perfora una banda (deve essere specificata l'informazione relativa all'idirizzo)
R — Leggi una banda (deve essere specificata l'informazione relativa all'indirizzo)

S — Passo - passo, esegui un'istruzione e digita i risultati

X — Prosegui nell'esecuzione del programma

K — Elimina il breakpoint

Se deve essere digitato un numero, DBUG si aspetta un indirizzo sotto forma di due parole ottali a tre digit o due parole esadecimali a due digit per un indirizzo HI e LO.

Dopo che è stato digitato un indirizzo (XXX YYY o XX YY)/ potete digitare B, G, L, /, CR o LF:

XXX YYY	B	Inserisci un breakpoint alla locazione specificata
	G	Inizia ad eseguire il programma alla locazione specificata
	L	Elenca le locazioni di memoria ed i loro contenuti in codice esadecimale o ottale, partendo dalla locazione di memoria specificata
	/	Metti in uscita il contenuto della locazione di memoria specificata
	CR	Esci sul command mode
	LF	Conserva la nuova parola dati (se specificata) e metti in uscita la locazione di memoria consecutiva ed il suo contenuto.

APPENDICE E

Listing ottale di DBUG

```
                        /   ***     ***    *****    ***    *
                        / *     *  *     *   *     *     *  *
                        / *     *  *         *     *     *  *
                        / *     *  *     *   *     *****    *
                        /   ***     ***      *     *     *  *****

                        /THIS IS A 1K VERSION OF TYCHON DBUG.
                        /VERSION 3, 1MAY77
                        /COPYRIGHT TYCHON INC., 1976

                          *000 070
000 070 303   VECT,       JMP       /WHEN THE RST7 INSTRUCTION IS EXECUTED
000 071 032               ENTRY     /WE VECTOR TO HERE AND THEN JUMP BACK
000 072 022               0         /TO DBUG.

                          *003 300
003 300 000   BRKADD,     000       /THIS IS THE ADDRESS OF THE USER
003 301 000               000       /INSERTED BREAKPOINT.
003 302 000   NEWADD,     000       /THIS IS THE ADDRESS OF THE NEXT EXECUT-
003 303 000               000       /ABLE INSTRUCTION AFTER THE BREAKPOINT.
003 304 000   TMP,        000       /THREE BYTES OF THE USER'S PROGRAM ARE
003 305 000               000       /DUPLICATED HERE IN RAM STARTING
003 306 000               000       /FROM THE BREAKPOINT ADDRESS.
003 307 303               JMP
003 310 323               EXECUT    /AFTER EXECUTING 1 INSTRUCTION, WE
003 311 022               0         /TYPE OUT THE REGISTERS ETC.
003 312 000   LAB,        000       /COUNTER FOR REGISTER "LABELS".
003 313 000   USERSK,     000       /THE USER'S STACK POINTER
003 314 000               000       /IS STORED HERE.
003 315 000   TEMPO,      000       /2 TEMPORARY STORAGE LOCATIONS
003 316 000               000
003 317 000   FLAG,       000       /THE A REGISTER AND FLAGS GO HERE.
003 320 000               000

                          *003 376
003 376 000   STACK,      000       /THE STACK POINTER IS SET FOR HERE.

                        /THIS IS THE STARTING ADDRESS OF DBUG !

                          *020 000
020 000 000   START,      NOP       /THESE 8 MEMORY LOCATIONS
020 001 000               NOP       /CAN BE USED TO INITIALIZE
020 002 000               NOP       /A USART
020 003 000               NOP
020 004 000               NOP
020 005 000               NOP
020 006 000               NOP
020 007 000               NOP
020 010 061               LXISP
020 011 376               STACK
020 012 003               0
020 013 021               LXID      /SET D&E = 000 070
020 014 070               VECT
```

```
020 015 000            0
020 016 041            LXIH       /SET H&L = STORAGE OF "JMP VECT"
020 017 361            BUFF2
020 020 023            0
020 021 315            CALL       /WRITE THE "JMP VECT" INTO RAM
020 022 247            SETUP      /STARTING FROM 000 070.
020 023 023            0
020 024 021            LXID       /D&E = 000 130
020 025 307            TMP+3
020 026 003            000
020 027 016            MVIC       /SET THE TRANSFER COUNTER TO 6
020 030 006            006        /FOR JMP(3), LAB(1) & USP(2)
020 031 315            CALL       /TRANSFER 6 BYTES FROM H&L TO D&E
020 032 251            SETUP+2    /SKIP THE MVIC 003 !
020 033 023            0
020 034 341  QUEST,    POPH       /DESTROY RETURN ADDRESS ON STACK
020 035 315            CALL
020 036 075            CRLF       /TYPE A CARRIAGE RETURN - LINE FEED
020 037 021            0
020 040 076            MVIA
020 041 277            277
020 042 315            CALL
020 043 336            TTYOUT     /THEN TYPE OUT A ?.
020 044 021            0
020 045 315  QUEST2    CALL       /TYPE A CARRIAGE RETURN - LINE FEED
020 046 075            CRLF
020 047 021            0
020 050 315  QUEST3,   CALL
020 051 331            TTYIN      /GET A KEYBOARD CHARACTER (USER COMMAND)
020 052 021            0
020 053 376            CPI
020 054 122            "R"        /R FOR READ A PAPER TAPE
020 055 302            JNZ
020 056 143            NOREAD     /NOT AN R, TRY A P
020 057 020            0
020 060 315  RDR,      CALL       /READ A CHARACTER
020 061 274            READ
020 062 021            0
020 063 376            CPI        /WAS THE CHARACTER READ IN
020 064 200            200        /LEADER (200) ?
020 065 312            JZ         /YES, GET ANOTHER CHARACTER
020 066 060            RDR
020 067 020            0
020 070 376  CHKFRM,   CPI        /NO, WAS IT AN ADDRESS FLAG ?
020 071 100            100        /(SEE THE PAPER TAPE FORMAT)
020 072 312            JZ         /YES, IT WAS AN ADDRESS FLAG
020 073 115            AIN
020 074 020            0
020 075 376            CPI        /WAS IT TRAILER (200) ?
020 076 200            200
020 077 312            JZ         /YES, RETURN TO THE COMMAND DECODER
020 100 045            QUEST2
020 101 020            0
020 102 315  NOEND,    CALL       /NO, IT MUST BE DATA TO BE SAVED
```

```
020 103 133             BYTE1       /COMBINE THE 2 BIT AND 6 BIT BYTE
020 104 020             0           /INTO AN 8 BIT BYTE
020 105 167             MOVMA       /SAVE IT IN MEMORY
020 106 043             INXH
020 107 315   NXTIN,    CALL        /AND GET ANOTHER PAPER TAPE FRAME.
020 110 274             READ
020 111 021             0
020 112 303             JMP
020 113 070             CHKFRM
020 114 020             0

020 115 315   AIN,      CALL        /AN ADDRESS FLAG WAS FOUND
020 116 130             BYTE        /THE NEXT 4 FRAMES REPRESENT
020 117 020             0           /THE H&L ADDRESS
020 120 147             MOVHA
020 121 315             CALL
020 122 130             BYTE
020 123 020             0
020 124 157             MOVLA       /SAVE THE ADDRESS IN H&L
020 125 303             JMP
020 126 107             NXTIN
020 127 020             0

                  /THE FORMAT OF THE PAPER TAPE IS, THE 2 MSB'S ARE
                  /PUNCHED OUT FIRST, THEN THE REMAINING 6 BITS
                  /ARE PUNCHED OUT.

020 130 315   BYTE,     CALL        /GET THE 2 MSB'S
020 131 274             READ
020 132 021             0
020 133 017   BYTE1,    RRC         /ROTATE THEM INTO THE MSB
020 134 017             RRC
020 135 117             MOVCA       /AND SAVE THEM IN C
020 136 315             CALL        /GET THE NEXT PAPER TAPE FRAME
020 137 274             READ
020 140 021             0
020 141 201             ADDC        /ADD THE MSB'S TO THE LSB'S
020 142 311             RET         /AND RETURN WITH THE 8 BIT # IN A.

020 143 376   NOREAD,   CPI
020 144 120             "P"         /P FOR PUNCH A PAPER TAPE
020 145 302             JNZ         /NOT A P, SEE IF IT WAS
020 146 267             NOPUN       /A VALID OCTAL (0-7) NUMBER BEING
020 147 020             0           /USED TO SPECIFY AN ADDRESS.
020 150 315   PUNCH,    CALL        /AFTER THE P WAS TYPED IN, PRINT
020 151 075             CRLF        /A CR & LF.
020 152 021             0
020 153 315             CALL        /GET THE BEGINNING OF FILE ADDRESS
020 154 116             TWOOCT
020 155 021             0
020 156 353             XCHG        /PUT IT IN H&L
020 157 315             CALL
020 160 075             CRLF
020 161 021             0
```

```
020 162 315            CALL      /GET THE END OF FILE ADDRESS
020 163 116            TWOOCT
020 164 021            0
020 165 173            MOVAE     /CALCULATE THE NUMBER OF BYTES
020 166 225            SUBL      /TO BE PUNCHED OUT.
020 167 137            MOVEA
020 170 172            MOVAD
020 171 234            SBBH
020 172 127            MOVDA
020 173 023            INXD      /AND SET THE RESULT TO 1 MORE.
020 174 315            CALL      /PUNCH OUT 10" OF LEADER
020 175 235            LDDR
020 176 020            0
020 177 076            MVIA      /PUNCH OUT AN ADDRESS FLAG
020 200 100            100
020 201 315            CALL
020 202 336            TTYOUT
020 203 021            0
020 204 174            MOVAH
020 205 315            CALL      /PUNCH OUT THE BEGINNING HI ADDRESS
020 206 251            BYTOUT    /AS A 2 & 6 BIT BYTE.
020 207 020            0
020 210 175            MOVAL
020 211 315            CALL      /PUNCH OUT THE BEGINNING LO ADDRESS
020 212 251            BYTOUT    /AS A 2 & 6 BIT BYTE.
020 213 020            0
020 214 176   NXTOUT,  MOVAM     /THEN START PUNCHING OUT THE
020 215 315            CALL      /CONTENTS OF MEMORY ADDRESSED BY H&L
020 216 251            BYTOUT
020 217 020            0
020 220 043            INXH
020 221 033            DCXD      /ANY MORE DATA TO BE PUNCHED OUT ?
020 222 172            MOVAD
020 223 263            ORAE
020 224 302            JNZ       /YES, GET THE NEXT MEMORY LOCATION
020 225 214            NXTOUT    /NO, THEN PUNCH OUT TRAILER AND
020 226 020            0         /RETURN TO THE COMMAND DECODER
020 227 315            CALL
020 230 235            LDDR
020 231 020            0
020 232 303            JMP
020 233 045            QUEST2
020 234 020            0
020 235 016   LDDR,    MVIC
020 236 144            144       /C=144=100 DECIMAL=10" OF LEADER
020 237 076            MVIA
020 240 200            200       /CODE FOR LEADER AND TRAILER
020 241 315            CALL
020 242 336            TTYOUT    /PUNCH THE 200 CODE
020 243 021            0
020 244 015            DCRC      /10" OF LEADER OR TRAILER YET ?
020 245 302            JNZ       /NO, PUNCH SOME MORE
020 246 237            LDDR+2
020 247 020            0
```

```
020 250 311           RET       /YES, RETURN

020 251 117 BYTOUT,   MOVCA     /DIVIDE AN 8 BIT BYTE INTO
020 252 346           ANI       /A 2 BIT AND A 6 BIT BYTE
020 253 300           300
020 254 007           RLC
020 255 007           RLC
020 256 315           CALL      /PUNCH OUT THE 2 MSB'S
020 257 336           TTYOUT
020 260 021           0
020 261 171           MOVAC
020 262 346           ANI       /THEN PUNCH OUT THE 6 LSB'S
020 263 077           077
020 264 303           JMP
020 265 336           TTYOUT
020 266 021           0

                      /A "K" TYPED IN MEANS TO REMOVE THE BREAKPOINT.
                      /A "X" TYPED IN MEANS TO CONTINUE EXECUTION
                      /A "S" TYPED IN MEANS TO SINGLE STEP

                      /SINCE THESE ARE COMMANDS WHICH REQUIRE THE
                      /USE OF THE BREAKPOINT, DBUG CHECKS TO SEE
                      /WHETHER THE USER HAS SET A BREAKPOINT
                      /SOMEPLACE. IF ONE HAS NOT BEEN SET, THEN
                      /WE CANNOT EXECUTE THESE COMMANDS.

020 267 137 NOPUN,    MOVEA     /SAVE THE CHARACTER IN E
020 270 376 BCS,      CPI       /S IS FOR SINGLE STEP
020 271 123           "S"
020 272 312           JZ        /IT WAS A S, EXECUTE 1 INSTRUCTION !
020 273 134           STEP
020 274 023           0
020 275 376           CPI       /X IS FOR CONTINUE (FULL SPEED)
020 276 130           "X"       /PROGRAM EXECUTION
020 277 312           JZ        /IT WAS A X, CONTINUE EXECUTION
020 300 074           CONTIN
020 301 023           0
020 302 376           CPI       /K IS FOR REMOVE THE LAST BREAKPOINT
020 303 113           "K"       /(THE USER HAS FOUND A BETTER PLACE)
020 304 302           JNZ       /NOT A K
020 305 331           NOBCS     /NOT S,K OR X, MAYBE ITS AN OCTAL NUMBER
020 306 020           0
020 307 052           LHLD      /A "K" WAS TYPED TO REMOVE A BREAKPOINT
020 310 300           BRKADD    /NOW SEE IF THERE IS ONE TO REMOVE !
020 311 003           0
020 312 176           MOVAM     /IS THERE A BREAKPOINT IN MEMORY
020 313 376           CPI       /TO BE REMOVED ?
020 314 377           377
020 315 302           JNZ       /NO, GO BACK TO THE COMMAND DECODER
020 316 035           QUEST+1   /SO IT'S NOT A VALID COMMAND.
020 317 020           0
020 320 315           CALL      /IT WAS A K, TAKE THE CONTENTS OF "TMP"
```

```
020 321 177          RSTRE      /AND PUT IT BACK IN THE USER'S PROGRAM
020 322 023          0          /USING "BRKADD" AS THE ADDRESS.
020 323 315          CALL       /SINCE THE BREAKPOINT HAS BEEN REMOVED
020 324 240          ZEROIT     /WE HAVE TO RESET BRKADD TO 377 377
020 325 023          0          /TO PREVENT THE USE OF K,X OR S.
020 326 303          JMP
020 327 045          QUEST2     /NOW GET ANOTHER COMMAND.
020 330 020          0
020 331 315  NOBCS,  CALL       /IT WAS A VALID OCTAL NUMBER, SO
020 332 107          SPCOCT     /CONVERT THE ASCII TO OCTAL
020 333 021          0          /(THERE MUST BE 3 DIGITS TYPED IN)
020 334 123          MOVDE      /SAVE THE HI ADDRESS IN D
020 335 315          CALL       /NOW GET THE 3 DIGIT LO ADDRESS
020 336 122          OCTIN      /ON RETURNING, IT IS IN E
020 337 021          0
020 340 353          XCHG       /THE ADDRESS IS NOW IN H&L
020 341 315          CALL       /NOW THAT AN ADDRESS HAS BEEN SPECIFIED,
020 342 331          TTYIN      /SEE WHAT THE USER WANTS TO DO AT
020 343 021          0          /THAT ADDRESS !
020 344 376          CPI
020 345 107          "G"        /G=GO, BEGINNING AT THE USER'S ADDRESS
020 346 302          JNZ        /A G WAS NOT TYPED IN, TRY A B
020 347 365          NOGO
020 350 020          0
020 351 315          CALL       /PRINT OUT A CR,LF AFTER THE G
020 352 075          CRLF       /WAS TYPED IN
020 353 021          0
020 354 257          XRAA       /SET A - E TO 000
020 355 107          MOVBA
020 356 117          MOVCA
020 357 127          MOVDA
020 360 137          MOVEA
020 361 345          PUSHH      /PUSH THE ADDRESS ON THE STACK
020 362 147          MOVHA      /THEN SET H&L TO 000
020 363 157          MOVLA
020 364 311          RET        /POP THE USER'S ADDRESS INTO THE
                                /PROGRAM COUNTER, AND OFF YOU GO.


020 365 376  NOGO,   CPI        /B AFTER AN ADDRESS = SET A BREAKPOINT.
020 366 102          "B"
020 367 312          JZ         /IT WAS A B, SO SET UP THE BREAKPOINT.
020 370 376          BREAK
020 371 021          0
020 372 376          CPI        /L=LIST THE CONTENTS OF MEMORY
020 373 114          "L"        /BEGINNING AT THE USER
020 374 312          JZ         /SPECIFIED ADDRESS.
020 375 200          LIST
020 376 021          0
020 377 376          CPI        /A "/" MEANS TYPE OUT THE
021 000 057          "/"        /CONTENTS OF MEMORY.
021 001 302          JNZ
021 002 035          QUEST+1    /NOT A G,B,L OR / SO TYPE OUT
021 003 020          0          /A ? AND GET ANOTHER COMMAND.
```

```
021 004 315            CALL     /WE TYPED A SLASH SO PUT OUT A SPACE
021 005 362            SPC
021 006 021            0
021 007 176  NXTLOC,   MOVAM    /GET THE CONTENTS OF MEMORY
021 010 315            CALL
021 011 234            OCTOUT   /PRINT IT OUT AS A 3
021 012 021            0        /DIGIT OCTAL NUMBER
021 013 315            CALL     /DID THE USER THAN TYPE IN A
021 014 317            TTYI     /CR, LF OR AN OCTAL # ?
021 015 021            0
021 016 346            ANI      /MASK OUT ANY PARITY BIT
021 017 177            177
021 020 376            CPI
021 021 015            015      /CR=RETURN TO THE COMMAND DECODER
021 022 312            JZ
021 023 045            QUEST2
021 024 020            0
021 025 376            CPI
021 026 012            012      /LF=CONTENTS OF THE NEXT MEMORY LOCATION
021 027 302            JNZ      /ANOTHER OCTAL NUMBER WAS TYPED IN
021 030 044            CNT      /(NEW CONTENTS OF THE SAME LOCATION)
021 031 021            0
021 032 315  NXTLN,    CALL     /IT WAS A LF, SO PRINT OUT THE
021 033 075            CRLF
021 034 021            0
021 035 043            INXH
021 036 315            CALL     /ADDRESS AND CONTENTS OF THE NEXT
021 037 352            HLSLSH   /CONSECUTIVE MEMORY LOCATION
021 040 021            0        /HLSLSH=H&L THEN A / AND SPACE.
021 041 303            JMP
021 042 007            NXTLOC   /NOW TYPE OUT THE CONTENTS OF
021 043 021            0        /THE NEXT LOCATION

021 044 315  CNT,      CALL     /OUTPUT THE FIRST CHARACTER (NUMBER)
021 045 336            TTYOUT
021 046 021            0
021 047 315            CALL     /GO TO THE OCTAL INPUT ROUTINE
021 050 107            SPCOCT
021 051 021            0
021 052 163            MOVME    /SAVE THE NUMBER INPUT
021 053 315            CALL     /SEE IF A CR, LF OR OCTAL
021 054 317            TTYI     /NUMBER WAS THEN TYPED IN
021 055 021            0
021 056 346            ANI      /MASK OUT ANY PARITY BIT
021 057 177            177
021 060 376            CPI
021 061 015            015      /CR=RETURN TO THE COMMAND DECODER
021 062 312            JZ
021 063 045            QUEST2
021 064 020            0
021 065 376            CPI
021 066 012            012
021 067 302            JNZ      /IT WASN'T A LF, IT MUST BE
021 070 044            CNT      /NEW CONTENTS AGAIN !
```

```
021 071 021              0
021 072 303              JMP
021 073 032              NXTLN        /A LF WAS TYPED IN, TYPE OUT THE
021 074 021              0            /NEXT MEM. LOCATION AND ITS CONTENTS

021 075 076   CRLF,      MVIA         /TYPE OUT A CR & LF
021 076 215              215
021 077 315              CALL
021 100 336              TTYOUT
021 101 021              0
021 102 076              MVIA
021 103 212              212
021 104 303              JMP
021 105 336              TTYOUT
021 106 021              0

                      /THIS IS THE OCTAL INPUT ROUTINE

021 107 036   SPCOCT,    MVIE         /E IS A TEMPORARY STORAGE REGISTER
021 110 000              000
021 111 016              MVIC         /C IS A DIGIT INPUT COUNTER
021 112 003              003
021 113 303              JMP
021 114 131              NXTOCT+3
021 115 021              0

021 116 315   TWOOCT,    CALL         /GET ONE THREE DIGIT OCTAL NUMBER
021 117 122              OCTIN
021 120 021              0
021 121 123              MOVDE        /SAVE THE VALUE IN "D"
021 122 257   OCTIN,     XRAA
021 123 137              MOVEA
021 124 016              MVIC
021 125 003              003
021 126 315   NXTOCT,    CALL         /GET A TTY CHARACTER
021 127 331              TTYIN
021 130 021              0
021 131 376              CPI          /AND SEE IF IT IS A VALID OCTAL #
021 132 060              060          /IF IT ISN'T, TYPE A ?
021 133 332              JC
021 134 034              QUEST
021 135 020              0
021 136 376              CPI
021 137 070              070
021 140 322              JNC
021 141 034              QUEST
021 142 020              0
021 143 346              ANI          /MASK OUT ALL BUT THE 3 LSB'S
021 144 007              007
021 145 107              MOVBA        /SAVE THE # IN B
021 146 173              MOVAE        /GET THE PREVIOUS# (INITIALLY = 0)
021 147 007              RLC          /ROTATE IT INTO THE 3 MIDDLE BITS.
021 150 007              RLC
021 151 007              RLC
```

```
021 152 200          ADDB      /ADD THE # JUST INPUT
021 153 137          MOVEA     /AND SAVE IT IN E
021 154 015          DCRC      /3 DIGITS YET ?
021 155 302          JNZ       /NO, GET ANOTHER 1
021 156 126          NXTOCT
021 157 021          0
021 160 303          JMP       /YES, THEN TYPE OUT A SPACE.
021 161 362          SPC
021 162 021          0
021 163 000          0
021 164 000          0
021 165 000          0
021 166 000          0
021 167 000          0
021 170 000          0
021 171 000          0
021 172 000          0
021 173 000          0
021 174 000          0
021 175 000          0
021 176 000          0
021 177 000          0

                       /L=LIST THE CONSECUTIVE MEMORY LOCATIONS
                       /AND THEIR CONTENTS UNTIL THE USER PRESSES
                       /A PRINTING TELETYPE KEY.

021 200 315   LIST,  CALL      /TYPE A CR & LF
021 201 075          CRLF
021 202 021          0
021 203 315          CALL      /THEN THE H&L ADDRESS, A SLASH
021 204 352          HLSLSH    /AND A SPACE
021 205 021          0
021 206 176          MOVAM     /GET THE CONTENTS OF MEMORY
021 207 315          CALL      /AND PRINT IT OUT.
021 210 234          OCTOUT
021 211 021          0
021 212 043          INXH      /INCREMENT THE MEMORY POINTER
021 213 333          IN        /ANY TELETYPE KEY PRESSED YET ?
021 214 021          021       /TTY SENSE REGISTER
021 215 346          ANI
021 216 001          001
021 217 312          JZ        /NO, LIST OUT THE NEXT LOCATION
021 220 200          LIST
021 221 021          0
021 222 333          IN        /YES, INPUT THE TERMINATING
021 223 020          020       /CHARACTER
021 224 303          JMP
021 225 045          QUEST2    /AND GO TO THE COMMAND DECODER.
021 226 020          0

                       /TYPE OUT THE CONTENTS OF H&L AS 2
                       /3 DIGIT OCTAL NUMBERS, WITH A SPACE AFTER
                       /THE HI AND LO ADDRESS.
```

```
021 227 174  HLOUT,   MOVAH
021 230 315           CALL
021 231 234           OCTOUT
021 232 021           0
021 233 175           MOVAL
021 234 117  OCTOUT,  MOVCA
021 235 346           ANI
021 236 300           300
021 237 007           RLC
021 240 007           RLC
021 241 306           ADI       /YOU ADD 260 TO MAKE IT AN ASCII #
021 242 260           260
021 243 315           CALL
021 244 336           TTYOUT
021 245 021           0
021 246 171           MOVAC
021 247 346           ANI
021 250 070           070
021 251 017           RRC
021 252 017           RRC
021 253 017           RRC
021 254 306           ADI
021 255 260           260
021 256 315           CALL
021 257 336           TTYOUT
021 260 021           0
021 261 171           MOVAC
021 262 346           ANI
021 263 007           007
021 264 306           ADI
021 265 260           260
021 266 315           CALL
021 267 336           TTYOUT
021 270 021           0
021 271 303           JMP
021 272 362           SPC
021 273 021           0

021 274 323  READ,    OUT       /PULSE THE READER CONTROL RELAY
021 275 021           021
021 276 000           NOP
021 277 000           NOP
021 300 000           NOP
021 301 000           NOP
021 302 000           NOP
021 303 000           NOP
021 304 000           NOP
021 305 000           NOP
021 306 000           NOP
021 307 000           NOP
021 310 000           NOP
021 311 000           NOP
021 312 000           NOP
```

```
021 313 000          NOP
021 314 000          NOP
021 315 000          NOP
021 316 000          NOP
021 317 333 TTYI,    IN         /HAS A CHARACTER BEEN RECEIVED YET ?
021 320 021          021
021 321 346          ANI
021 322 001          001
021 323 312          JZ         /NO, KEEP WAITING FOR THE FLAG
021 324 317          TTYI
021 325 021          0
021 326 333          IN         /YES, THEN INPUT IT
021 327 020          020
021 330 311          RET

021 331 315 TTYIN,   CALL       /GET A TELETYPE CHARACTER
021 332 317          TTYI
021 333 021          0
021 334 346          ANI        /AND THEN ECHO IT
021 335 177          177
021 336 107 TTYOUT,  MOVBA      /SAVE THE CHARACTER TO BE PRINTED IN B
021 337 333          IN         /IS THE TRANSMITTER READY YET ?
021 340 021          021
021 341 346          ANI
021 342 004          004
021 343 312          JZ         /NO, KEEP WAITING
021 344 337          TTYOUT+1
021 345 021          0
021 346 170          MOVAB      /YES, GET THE CHARACTER
021 347 323          OUT        /AND TRANSMIT IT.
021 350 020          020
021 351 311          RET

021 352 315 HLSLSH,  CALL
021 353 227          HLOUT      /TYPE OUT H & L, THEN A SPACE
021 354 021          0
021 355 076          MVIA       /THEN TYPE A "/"
021 356 257          257
021 357 315          CALL
021 360 336          TTYOUT
021 361 021          0

                     /PRINT OUT SPACES, THE NUMBER OF WHICH
                     /IS STORED IN C.

021 362 016 SPC,     MVIC
021 363 001          001
021 364 076 MORSPC,  MVIA
021 365 240          240        /ASCII SPACE
021 366 315          CALL
021 367 336          TTYOUT
021 370 021          0
021 371 015          DCRC       /ANYMORE TO PRINT ?
021 372 302          JNZ        /YES, PRINT ANOTHER 1
```

```
021 373 364           MORSPC
021 374 021           0
021 375 311           RET        /NO, THEN RETURN

021 376 315  BREAK,   CALL       /THE USER WANTS TO PUT A BREAKPOINT
021 377 004           BRK        /IN A PROGRAM
022 000 022           0
022 001 303           JMP
022 002 045           QUEST2
022 003 020           0

022 004 042  BRK,     SHLD       /SAVE THE ADDRESS OF THE BREAK-
022 005 300           BRKADD     /POINT ADDRESS IN BRKADD.
022 006 003           0
022 007 353           XCHG       /NOW WE WANT TO DUPLICATE PART OF
022 010 041           LXIH       /USER'S PROGRAM, STARTING A THE
022 011 304           TMP        /BREAKPOINT ADDRESS, DOWN AT "TMP"
022 012 003           0
022 013 032  BREAK1,  LDAXD
022 014 107           MOVBA
022 015 076           MVIA       /WRITE A RST7 INTO THE USER'S PROGRAM
022 016 377           377        /THIS IS A RST7 INSTRUCTION
022 017 022           STAXD
022 020 160           MOVMB
022 021 043           INXH
022 022 023           INXD
022 023 032           LDAXD
022 024 167           MOVMA
022 025 043           INXH
022 026 023           INXD
022 027 032           LDAXD
022 030 167           MOVMA
022 031 311           RET        /WE HAVE DONE 3 MEMORY LOCATIONS

                     /WE "HIT" THE RST7 IN THE USER'S PROGRAM AND
                     /WE GET TO ENTRY BY THE JUMP INSTRUCTION STORED
                     /AT LOC. 000 070. THE RST7 MAY BE CHANGED
                     /AT THIS POINT, ALL OF THE REGISTERS
                     /AND THE STACK HAVE USER VALUES IN THEM
                     /SO DBUG HAS TO PRESERVE THEM.

022 032 042  ENTRY,   SHLD       /SAVE H&L IN "TEMPO"
022 033 315           TEMPO
022 034 003           0
022 035 365           PUSHPSW
022 036 341           POPH       /GET THE FLAGS AND A IN H&L
022 037 042           SHLD       /NOW SAVE THEM IN "FLAG"
022 040 317           FLAG
022 041 003           0
022 042 041           LXIH       /NOW DESTROY THE RETURN ON THE STACK
022 043 002           002        /DUE TO THE RST INSTRUCTION AND
022 044 000           000        /GET THE STACK POINTER IN H&L
022 045 071  SENTRY,  DADSP
022 046 042           SHLD       /SAVE THE USER'S STACK IN "USERSK"
```

```
022 047 313        USERSK
022 050 003        0
022 051 061        LXISP      /NOW SET THE SP SO DBUG CAN USE IT
022 052 376        STACK
022 053 003        0
022 054 052        LHLD       /GET H&L BACK INTO H&L FROM "TEMPO"
022 055 315        TEMPO
022 056 003        0
022 057 345        PUSHH      /PUSH B - H ONTO DBUG'S STACK
022 060 325        PUSHD
022 061 305        PUSHB
022 062 052        LHLD       /NOW GET THE FLAGS AND A INTO H&L
022 063 317        FLAG
022 064 003        0
022 065 345        PUSHH      /AND PUSH THEM ON THE STACK.
022 066 052        LHLD       /GET THE USER SELECTED BREAKPOINT
022 067 300        BRKADD     /ADDRESS AND TYPE IT OUT SO WE DON'T
022 070 003        0          /FORGET WHAT IT WAS.
022 071 315        CALL
022 072 227        HLOUT
022 073 021        0
022 074 072        LDA        /GET THE USER INSTRUCTION AT "TMP"
022 075 304        TMP
022 076 003        0
022 077 107        MOVBA      /AND SEE IF IT IS EXECUTABLE.
022 100 376        CPI
022 101 303        303        /CAN'T DO A JUMP
022 102 312        JZ
022 103 247        NOGOOD
022 104 022        0
022 105 376        CPI
022 106 315        315        /CAN'T CALL ANY SUBROUTINES
022 107 312        JZ
022 110 247        NOGOOD
022 111 022        0
022 112 376        CPI
022 113 311        311        /RETURN
022 114 312        JZ
022 115 247        NOGOOD
022 116 022        0
022 117 376        CPI
022 120 351        351        /PCHL
022 121 312        JZ
022 122 247        NOGOOD
022 123 022        0
022 124 376        CPI
022 125 333        333        /OUTPUT INSTRUCTION IS A 2 BYTE INSTR.
022 126 312        JZ
022 127 235        IMMED
022 130 022        0
022 131 376        CPI
022 132 323        323        /AND SO ARE INPUT INSTRUCTIONS.
022 133 312        JZ
022 134 235        IMMED
```

```
022 135 022            0
022 136 346  NO300,    ANI       /IT WASN'T 1 OF THOSE, TRY THESE.
022 137 307            307
022 140 376            CPI
022 141 306            306
022 142 312            JZ
022 143 235            IMMED
022 144 022            0
022 145 376            CPI
022 146 302            302       /CONDITIONAL JUMPS
022 147 312            JZ
022 150 247            NOGOOD
022 151 022            0
022 152 376            CPI
022 153 304            304       /CONDITIONAL CALLS
022 154 312            JZ
022 155 247            NOGOOD
022 156 022            0
022 157 376            CPI
022 160 300            300       /CONDITIONAL RETURNS
022 161 312            JZ
022 162 247            NOGOOD
002 163 022            0
022 164 376            CPI
022 165 307            307       /RESTARTS
022 166 312            JZ
022 167 247            NOGOOD
022 170 022            0
022 171 376            CPI
022 172 006            006       /IMMEDIATE MOVES
022 173 312            JZ
022 174 235            IMMED
022 175 022            0
022 176 376            CPI
022 177 001            001       /LOAD IMMEDIATE REGISTER PAIR
022 200 312            JZ
022 201 224            IMMSL1
022 202 022            0
022 203 376            CPI
022 204 002            002       /STA, LDA, SHLD OR LHLD
022 205 312            JZ
022 206 263            IMMSL2
022 207 022            0
022 210 041  SING,     LXIH      /IT WAS A SINGLE BYTE INSTRUCTION
022 211 000            000       /SO WE WRITE 2 NOPS IMMEDIATELY
022 212 000            000       /AFTER THE INSTRUCTION.
022 213 042            SHLD
022 214 305            TMP+1
022 215 003            0
022 216 052            LHLD      /SET H&L = TO THE CONTENTS OF "BRKADD"
022 217 300            BRKADD
022 220 003            0
022 221 303            JMP
022 222 276            ONEMOR
```

```
022 223 022            0
022 224 170  IMMSL1,   MOVAB     /IS THE INSTRUCTIONS REALLY A
022 225 346            ANI       /SINGLE BYTE OR 3 BYTE ?
022 226 010            010
022 227 302            JNZ       /IF A=010, ITS A SINGLE BYTE
022 230 210            SING
022 231 022            0
022 232 303            JMP       /IF A=000, ITS A 3 BYTE
022 233 271            THREBY
022 234 022            0
022 235 257  IMMED,    XRAA      /IMMEDIATE AND I-O INSTRUCTIONS
022 236 062            STA       /ARE TWO BYTE INSTRUCTIONS
022 237 306            TMP+2     /SO WE SAVE 1 NOP AFTER THE
022 240 003            0         /INSTRUCTION.
022 241 052            LHLD
022 242 300            BRKADD
022 243 003            0
022 244 303            JMP
022 245 275            TWOMOR
022 246 022            0

022 247 052  NOGOOD,   LHLD      /IF THE BREAKPOINT WAS PLACED ON A
022 250 300            BRKADD    /NONEXECUTABLE INSTRUCTION, WE RE-
022 251 003            0         /MOVE THE RST7 FROM THE USER'S
022 252 042            SHLD      /PROGRAM AND GO TO THE COMMAND
022 253 302            NEWADD    /DECODER
022 254 003            0
022 255 315            CALL
022 256 177            RSTRE
022 257 023            0
022 260 303            JMP
022 261 040            QUEST+4
022 262 020            0
022 263 170  IMMSL2,   MOVAB     /IS IT REALLY A STA,LDA,SHLD OR LHLD ?
022 264 376            CPI
022 265 042            042
022 266 332            JC        /NO, THEN IT MUST BE A SINGLE BYTE.
022 267 210            SING
022 270 022            0
022 271 052  THREBY,   LHLD
022 272 300            BRKADD
022 273 003            0
022 274 043            INXH
022 275 043  TWOMOR,   INXH
022 276 043  ONEMOR,   INXH
022 277 042            SHLD      /STORE THE ADDRESS OF THE NEXT
022 300 302            NEWADD    /INSTRUCTION
022 301 003            0
022 302 361            POPPSW    /GET BACK ALL THE USER'S REGISTERS
022 303 301            POPB
022 304 321            POPD
022 305 341            POPH
022 306 042            SHLD      /SAVE H&L IN "TEMPO"
022 307 315            TEMPO
```

```
022 310 003           0
022 311 052           LHLD      /SET H&L WITH THE USER'S STACK POINTER
022 312 313           USERSK
022 313 003           0
022 314 371           SPHL      /NOW SET THE SP WITH THAT VALUE
022 315 052           LHLD      /NOW GET H&L BACK TO WHAT THEY WERE
022 316 315           TEMPO
022 317 003           0
022 320 303           JMP       /NOW DO THE 1,2 OR 3 BYTE INSTRUCTION
022 321 304           TMP       /STORED AT "TMP" (WE MAY ALSO DO
022 322 003           0         /1 OR 2 NOP'S).

                      /AFTER EXECUTING THE INSTRUCTION AT "TMP"
                      /WE JUMP BACK TO HERE. THE CONTENTS OF
                      /THE REGISTERS MUST BE SAVED AGAIN, SP'S
                      /SWITCHED ETC. BEFORE WE CAN TYPE OUT THE
                      /CONTENTS OF THE REGISTERS.

022 323 042  EXECUT,  SHLD      /FROM "TMP" WE JUMP TO HERE.
022 324 315           TEMPO     /SAVE H&L IN "TEMPO"
022 325 003           0
022 326 365           PUSHPSW
022 327 341           POPH
022 330 042           SHLD      /SAVE THE FLAGS AND A IN "FLAG"
022 331 317           FLAG
022 332 003           0
022 333 041           LXIH      /SET H&L = TO 000
022 334 000           0
022 335 000           0
022 336 071           DADSP     /PUT THE USER'S SP INTO H&L
022 337 042           SHLD      /AND SAVE IT IN "USERSK"
022 340 313           USERSK
022 341 003           0
022 342 061           LXISP     /SET UP A SP FOR DBUG TO USE
022 343 376           STACK
022 344 003           0
022 345 052           LHLD      /SET H&L TO WHAT THEY WHERE
022 346 315           TEMPO
022 347 003           0
022 350 345           PUSHH     /SAVE B-H ON THE STACK
022 351 325           PUSHD
022 352 305           PUSHB
022 353 052           LHLD      /GET THE FLAGS AND A INTO H&L
022 354 317           FLAG
022 355 003           0
022 356 345           PUSHH     /AND PUT THEM ON THE STACK ALSO.
022 357 315           CALL      /TYPE OUT A CR & LF
022 360 075           CRLF
022 361 021           0
022 362 041           LXIH      /DECREMENT THE REGISTER LABEL COUNTER
022 363 312           LAB
022 364 003           0
022 365 065           DCRM
022 366 314           CZ        /IF 0, TYPE OUT THE REGISTER LABELS
```

```
022 367 262          LABEL
022 370 023          0
022 371 041          LXIH
022 372 000          000
022 373 000          000
022 374 071          DADSP     /SET H&L = TO THE SP
022 375 136          MOVEM     /GET THE FLAG WORD
022 376 315          CALL      /AND DISSASSEMBLE IT INTO 1'S & 0'S
022 377 207          BIT
023 000 023          0
023 001 043          INXH
023 002 176          MOVAM     /NOW GET THE A REGISTER
023 003 315          CALL      /AND TYPE IT OUT AS A 3 DIGIT OCTAL #
023 004 234          OCTOUT
023 005 021          0
023 006 026          MVID      /NOW WE GET THE OTHER REGISTERS
023 007 003          003       /AND TYPE OUT THEIR OCTAL VALUES
023 010 036 TWOTIM,  MVIE
023 011 002          002
023 012 043          INXH
023 013 043          INXH
023 014 176 MOR1,    MOVAM
023 015 315          CALL
023 016 234          OCTOUT
023 017 021          0
023 020 053          DCXH
023 021 035          DCRE
023 022 302          JNZ
023 023 014          MOR1
023 024 023          0
023 025 043          INXH
023 026 043          INXH
023 027 025          DCRD      /ANYMORE REGISTER PAIRS ?
023 030 302          JNZ       /YES, DO ANOTHER
023 031 010          TWOTIM
023 032 023          0
023 033 126          MOVDM     /NO, GET THE VALUES OF H&L INTO H&L
023 034 053          DCXH
023 035 136          MOVEM
023 036 353          XCHG
023 037 176          MOVAM     /GET CNTS OF MEMORY ADDRESSED BY H & L
023 040 315          CALL      /AND TYPE OUT ITS VALUE
023 041 234          OCTOUT
023 042 021          0
023 043 052          LHLD      /SET H&L = TO THE USER'S SP
023 044 313          USERSK
023 045 003          0
023 046 315          CALL      /AND TYPE OUT THE 2 OCTAL WORDS
023 047 227          HLOUT
023 050 021          0
023 051 043          INXH      /GET THE HI OFF THE STACK
023 052 176          MOVAM
023 053 315          CALL      /AND PRINT IT OUT
023 054 234          OCTOUT
```

```
023 055 021            0
023 056 053            DCXH
023 057 176            MOVAM     /THEN GET THE LO OFF THE STACK
023 060 315            CALL      /AND PRINT IT OUT
023 061 234            OCTOUT
023 062 021            0
023 063 315            CALL      /PRINT A CR & LF AFTER ALL OF THIS
023 064 075            CRLF
023 065 021            0
023 066 315            CALL      /REMOVE THE BREAKPOINT (RST7)
023 067 177            RSTRE     /FROM THE USER'S PROGRAM
023 070 023            0
023 071 303            JMP       /AND GO BACK TO THE COMMAND DECODER
023 072 045            QUEST2
023 073 020            0

                      /IF WE TYPE A X TO CONTINUE, WE HAVE TO GET ALL
                      /OF THE REGISTERS BACK THE WAY THEY WERE, WITH
                      /THE VALUES IN THEM THAT THE USER ESTABLISHED, AL-
                      /ONG WITH THE USER'S STACK POINTER, AND
                      /THEN WE CAN USE THE CONTENTS OF "NEWADD"
                      /AS THE ADDRESS FROM WHERE WE SHOULD CONTINUE THE
                      /PROGRAMS EXECUTION.

023 074 052   CONTIN,  LHLD
023 075 300            BRKADD    /IS THERE A BREAKPOINT ADDRESS ?
023 076 003            0
023 077 043            INXH
023 100 174            MOVAH
023 101 265            ORAL
023 102 312            JZ        /NO, BRKADD WAS SET TO 377 377
023 103 035            QUEST+1
023 104 020            0
023 105 315            CALL      /THERE IS AN ADDRESS, CONTINUE
023 106 075            CRLF
023 107 021            0
023 110 361   CNTIN1,  POPPSW    /GET BACK A-H
023 111 301            POPB
023 112 321            POPD
023 113 341            POPH
023 114 042            SHLD      /SAVE H&L IN "TEMPO"
023 115 315            TEMPO
023 116 003            0
023 117 052            LHLD      /GET THE USER'S SP
023 120 313            USERSK
023 121 003            0
023 122 371            SPHL      /AND PUT IT INTO THE SP
023 123 052            LHLD      /GET H&L BACK THE WAY THEY SHOULD BE
023 124 315            TEMPO
023 125 003            0
023 126 345            PUSHH     /PUT H&L ON THE STACK
023 127 052            LHLD      /GET THE ADDRESS WHERE WE SHOULD
023 130 302            NEWADD    /CONTINUE PROGRAM EXECUTION
023 131 003            0
```

```
023 132 343          XTHL      /NEW ADDRESS ON STACK, H&L THE
023 133 311          RET       /WAY THEY WERE. HERE WE GO . . . .

                     /IF WE TYPE S FOR SINGLE STEP, WE WANT TO EXECUTE
                     /1 INSTRUCTION AND THEN SEE WHAT EFFECT IT HAD
                     /ON THE REGISTERS, STACK POINTER, STACK OR MEMORY.

023 134 052  STEP,   LHLD      /SEE IF THERE IS A BREAKPOINT
023 135 300          BRKADD    /ALREADY SET.
023 136 003          0
023 137 043          INXH
023 140 174          MOVAH
023 141 265          ORAL
023 142 312          JZ        /IF NOT BREAKPOINT, H=L=377
023 143 035          QUEST+1   /AFTER INXH, H=L=000 AND WE ERROR
023 144 020          0
023 145 315          CALL      /TYPE A SPACE AFTER THE S
023 146 362          SPC
023 147 021          0
023 150 052          LHLD      /GET THE ADDRESS OF THE NEXT EX-
023 151 302          NEWADD    /ECUTABLE INSTRUCTION.
023 152 003          0
023 153 315          CALL      /SET A BREAKPOINT AT THIS NEW ADDRESS
023 154 004          BRK
023 155 022          0
023 156 361          POPPSW    /GET THE REGISTERS AND SP BACK
023 157 301          POPB      /TO WHAT THEY SHOULD BE
023 160 321          POPD
023 161 341          POPH
023 162 042          SHLD
023 163 315          TEMPO
023 164 003          0
023 165 052          LHLD
023 166 313          USERSK
023 167 003          0
023 170 371          SPHL
023 171 041          LXIH
023 172 000          0
023 173 000          0
023 174 303          JMP       /THEN JUMP TO THE BEGINNING OF
023 175 045          SENTRY    /BREAKPOINT ROUTINE
023 176 022          0

023 177 072  RSTRE,  LDA       /GET THE FIRST BYTE OF THE INSTRUCTION
023 200 304          TMP
023 201 003          0
023 202 052          LHLD      /GET THE ADDRESS OF WHERE IT CAME FROM
023 203 300          BRKADD
023 204 003          0
023 205 167          MOVMA     /PUT IT BACK IN THE USER'S PROGRAM
023 206 311          RET       /WRITING OVER THE RST7 INSTRUCTION !

                     /THIS SUBROUTINE IS USED TO UNPACK THE FLAG
                     /REGISTER INTO A STRING OF 8 1'S AND 0'S.
```

```
                        /ENTER WITH THE WORD TO BE UNPACKED IN E

023 207 016  BIT,       MVIC     /C= THE ROTATE OR BIT COUNTER
023 210 010             010
023 211 173  NXTBIT,    MOVAE    /GET THE WORD
023 212 007             RLC      /ROTATE 1 BIT LEFT INTO THE CARRY
023 213 137             MOVEA    /AND SAVE THE WORD AGAIN
023 214 332             JC       /IF THE CARRY IS SET, WE ROTATED
023 215 233             ONE      /A 1 INTO IT, SO WE PRINT A 1.
023 216 023             0
023 217 076             MVIA     /OTHERWISE, WE TYPE A 0
023 220 260             260
023 221 315  ONEOUT,    CALL     /PRINT THE CHARACTER
023 222 336             TTYOUT
023 223 021             0
023 224 015             DCRC     /8 BITS YET ?
023 225 302             JNZ      /NO, DO ANOTHER ONE
023 226 211             NXTBIT
023 227 023             0
023 230 303             JMP      /YES, NOW PRINT A SPACE
023 231 362             SPC
023 232 021             0
023 233 076  ONE,       MVIA
023 234 261             261
023 235 303             JMP
023 236 221             ONEOUT
023 237 023             0

                        /THIS CLEARS THE CONTENTS OF "BRKADD"

023 240 041  ZEROIT,    LXIH
023 241 377             377      /SET BRKADD TO 377 377
023 242 377             377
023 243 042             SHLD
023 244 300             BRKADD
023 245 003             0
023 246 311             RET

                        /THIS DUPLICATES PORTIONS OF "BUFF2"
                        /INTO R-W MEMORY

023 247 016  SETUP,     MVIC
023 250 003             003
023 251 176             MOVAM
023 252 022             STAXD
023 253 043             INXH
023 254 023             INXD
023 255 015             DCRC
023 256 302             JNZ
023 257 251             SETUP+2
023 260 023             0
023 261 311             RET
```

```
                        /IF WE HAD DECREMENTED "LAB" TO 0, WE COME HERE.
                        /WE USE THE CONTENTS OF "REGLST"
                        /TO PRODUCE THE REGISTER LABELS. NOTICE THAT ANY
                        /NUMBER IN "REGLST" LESS THAN 010 (OCTAL), PRODUCES
                        /THAT NUMBER OF SPACES, EXCEPT 0, WHICH TERMINATES
                        /THE OUTPUT.

023 262 066  LABEL,     MVIM      /SET THE LABEL COUNTER TO 5 AGAIN
023 263 005             005
023 264 041             LXIH
023 265 316             REGLST    /H&L POINT TO "REGLST"
023 266 023             0
023 267 176  NXTLET,    MOVAM     /GET A "REGLST" TABLE CHARACTER
023 270 376             CPI       /IS IT A 000 ? (END OF THE TABLE ?)
023 271 000             0
023 272 310             RZ        /YES, THEN RETURN
023 273 376             CPI       /IS IT A NUMBER BELOW 010 ?
023 274 010             010       /IF SO, USE IT AS A SPACE COUNT
023 275 332             JC        /YES, SO TYPE SOME SPACES
023 276 307             SPCIT
023 277 023             0
023 300 315             CALL      /NONE OF THE ABOVE, JUST
023 301 336             TTYOUT    /PRINT THE CHARACTER.
023 302 021             0
023 303 043  NXTLAB,    INXH      /INCREMENT REGISTER PAIR H&L
023 304 303             JMP       /THEN GET AND INTERPRETE THE NEXT
023 305 267             NXTLET    /CHARACTER
023 306 023             0
023 307 117  SPCIT,     MOVCA
023 310 315             CALL
023 311 364             MORSPC
023 312 021             0
023 313 303             JMP
023 314 303             NXTLAB
023 315 023             0
023 316 323  REGLST,    323       /S
023 317 332             332       /Z
023 320 001             001       /1 SPACE
023 321 261             261       /1
023 322 001             001       /1 SPACE
023 323 320             320       /P
023 324 001             001       /1 SPACE
023 325 262             262       /2 (4 BIT CARRY)
023 326 002             002       /2 SPACES
023 327 301             301       /A
023 330 003             003       /3 SPACES
023 331 302             302       /B
023 332 003             003       /3 SPACES
023 333 303             303       /C
023 334 003             003       /3 SPACES
023 335 304             304       /D
023 336 003             003       /3 SPACES
023 337 305             305       /E
023 340 003             003       /3 SPACES
```

```
13 E1  C8            310      /H
13 E2  02            002      /2 SPACES
13 E3  CC            314      /L
13 E4  02            002      /2SPACES
13 E5  CD            315      /M
13 E6  03            003      /3 SPACES
13 E7  D3            323      /S
13 E8  01            001      /1 SPACE
13 E9  D0            320      /P
13 EA  03            003      /3 SPACES
13 EB  C3            303      /C
13 EC  01            001      /1 SPACE
13 ED  D3            323      /S
13 EE  8D            215      /CR
13 EF  8A            212      /LF
13 F0  00            000      /MESSAGE TERMINATOR
13 F1  C3   BUFF2,   JMP      /THIS IS WHAT GETS DUPLICATED
13 F2  1A            ENTRY    /IN RAM. THIS STARTS AT 000 070
13 F3  12            0
13 F4  C3            JMP      /THIS GOES TO 03 C7
13 F5  D3            EXECUT
13 F6  12            0
13 F7  01            001      /THIS IS THE VALUE OF "LAB"
13 F8  80            200
13 F9  03            003
```

```
AIN      =020 115  BRKADD  =003 300  BYTE    =020 130  BYTE1   =020 133
BYTOUT   =020 251  BCS     =020 270  BREAK   =021 376  BRK     =022 004
BREAK1   =022 013  BIT     =023 207  BUFF2   =023 361  CHKFRM  =020 070
CNT      =021 044  CRLF    =021 075  CONTIN  =023 074  CNTIN1  =023 110
ENTRY    =022 032  EXECUT  =022 323  FLAG    =003 317  HLOUT   =021 227
HLSLSH   =021 352  IMMSL1  =022 224  IMMED   =022 235  IMMSL2  =022 263
LAB      =003 312  LDDR    =020 235  LIST    =021 200  LABEL   =023 262
MORSPC   =021 364  MORI    =023 014  NEWADD  =003 302  NOEND   =020 102
NXTIN    =020 107  NOREAD  =020 143  NXTOUT  =020 214  NOPUN   =020 267
NOBCS    =020 331  NOGO    =020 365  NXTLOC  =021 007  NXTLN   =021 032
NXTOCT   =021 126  NO300   =022 136  NOGOOD  =022 247  NXTBIT  =023 211
NXTLET   =023 267  NXTLAB  =023 303  OCTIN   =021 122  OCTOUT  =021 234
ONEMOR   =022 276  ONEOUT  =023 221  ONE     =023 233  PUNCH   =020 150
QUEST    =020 034  QUEST2  =020 045  QUEST3  =020 050  RDR     =020 060
READ     =021 274  RSTRE   =023 177  REGLST  =023 316  STACK   =003 376
START    =020 000  SPCOCT  =021 107  SPC     =021 362  SENTRY  =022 045
SING     =022 210  STEP    =023 134  SETUP   =023 247  SPCIT   =023 307
TMP      =003 304  TEMPO   =003 315  TWOOCT  =021 116  TTYI    =021 317
TTYIN    =021 331  TTYOUT  =021 336  THREBY  =022 271  TWOMOR  =022 275
TWOTIM   =023 010  USERSK  =003 313  VECT    =000 070  ZEROIT  =023 240

ERRORS DETECTED = 000
```

APPENDICE F

Listing esadecimale di DBUG

```
                        / *      *  *****  *     *
                        / *      *  *        *  *
                        / ******  ***         **
                        / *      *  *        *  *
                        / *      *  *****  *     *

                        /THIS IS A 1K VERSION OF TYCHON DBUG.
                        /VERSION 3, 1MAY77
                        /COPYRIGHT TYCHON INC., 1977

                         *00H 38H
00 38  C3  VECT,        JMP      /WHEN THE RST7 INSTRUCTION IS EXECUTED
00 39  1A               ENTRY    /WE VECTOR TO HERE AND THEN JUMP BACK
00 3A  12               0        /TO DBUG.
                         *03H C0H
03 C0  00  BRKADD,      000      /THIS IS THE ADDRESS OF THE USER
03 C1  00               000      /INSERTED BREAKPOINT.
03 C2  00  NEWADD,      000      /THIS IS THE ADDRESS OF THE NEXT EXECUT-
03 C3  00               000      /ABLE INSTRUCTION AFTER THE BREAKPOINT.
03 C4  00  TMP,         000      /THREE BYTES OF THE USER'S PROGRAM ARE
03 C5  00               000      /DUPLICATED HERE IN RAM STARTING
03 C6  00               000      /FROM THE BREAKPOINT ADDRESS.
03 C7  C3               JMP
03 C8  D3               EXECUT   /AFTER EXECUTING 1 INSTRUCTION, WE
03 C9  12               0        /TYPE OUT THE REGISTERS ETC.
03 CA  00  LAB,         000      /COUNTER FOR REGISTER "LABELS"
03 CB  00  USERSK,      000      /THE USER'S STACK POINTER
03 CC  00               000      /IS STORED HERE.
03 CD  00  TEMPO,       000      /2 TEMPORARY STORAGE LOCATIONS
03 CE  00               000
03 CF  00  FLAG,        000      /THE  A REGISTER AND FLAGS GO HERE.
03 D0  00               000

                         *03H FEH
03 FE  00  STACK,       000      /THE STACK POINTER IS SET FOR HERE.

                        /THIS IS THE STARTING ADDRESS OF DBUG !

                         *10H 00H
10 00  00  START,       NOP      /THESE 8 MEMORY LOCATIONS
10 01  00               NOP      /CAN BE USED TO INITIALIZE
10 02  00               NOP      /A USART
10 03  00               NOP
10 04  00               NOP
10 05  00               NOP
10 06  00               NOP
10 07  00               NOP
10 08  31               LXISP
10 09  FE               STACK
10 0A  03               0
10 0B  11               LXID     /SET D&E = 00 38
10 0C  38               VECT
```

```
10 0D  00               0
10 0E  21               LXIH       /SET H&L = STORAGE OF "JMP VECT"
10 0F  F1               BUFF2
10 10  13               0
10 11  CD               CALL       /WRITE THE "JMP VECT" INTO RAM
10 12  A7               SETUP      /STARTING FROM 00 38.
10 13  13               0
10 14  11               LXID       /D&E = 03 C7
10 15  C7               TMP+3
10 16  03               000
10 17  0E               MVIC       /SET THE TRANSFER COUNTER TO 6
10 18  06               006        /FOR JMP(3), LAB(1) & USP(2)
10 19  CD               CALL       /TRANSFER 6 BYTES FROM H&L TO D&E
10 1A  A9               SETUP+2    /SKIP THE MVIC 003 !
10 1B  13               0
10 1C  E1    QUEST,     POPH       /DESTROY RETURN ADDRESS ON STACK
10 1D  CD               CALL
10 1E  3D               CRLF       /TYPE A CARRIAGE RETURN - LINE FEED
10 1F  11               0
10 20  3E               MVIA
10 21  BF               277
10 22  CD               CALL
10 23  DE               TTYOUT     /THEN TYPE OUT A ?.
10 24  11               0
10 25  CD    QUEST2,    CALL       /TYPE A CARRIAGE RETURN - LINE FEED
10 26  3D               CRLF
10 27  11               0
10 28  CD    QUEST3,    CALL
10 29  D9               TTYIN      /GET A KEYBOARD CHARACTER (USER COMMAND)
10 2A  11               0
10 2B  FE               CPI
10 2C  52               "R"        /R FOR READ A PAPER TAPE
10 2D  C2               JNZ
10 2E  63               NOREAD     /NOT AN R, TRY A P
10 2F  10               0
10 30  CD    RDR,       CALL       /READ A CHARACTER
10 31  BC               READ
10 32  11               0
10 33  FE               CPI        /WAS THE CHARACTER READ IN
10 34  80               200        /LEADER (200) ?
10 35  CA               JZ         /YES, GET ANOTHER CHARACTER
10 36  30               RDR
10 37  10               0
10 38  FE    CHKFRM,    CPI        /NO, WAS IT AN ADDRESS FLAG ?
10 39  40               100        /(SEE THE PAPER TAPE FORMAT)
10 3A  CA               JZ         /YES, IT WAS AN ADDRESS FLAG
10 3B  4D               AIN
10 3C  10               0
10 3D  FE               CPI        /WAS IT TRAILER (200) ?
10 3E  80               200
10 3F  CA               JZ         /YES, RETURN TO THE COMMAND DECODER
10 40  25               QUEST2
10 41  10               0
10 42  CD    NOEND,     CALL       /NO, IT MUST BE DATA TO BE SAVED
```

```
10 43  5B          BYTE1     /COMBINE THE 2 BIT AND 6 BIT BYTE
10 44  10          0         /INTO AN 8 BIT BYTE
10 45  77          MOVMA     /SAVE IT IN MEMORY
10 46  23          INXH
10 47  CD  NXTIN,  CALL      /AND GET ANOTHER PAPER TAPE FRAME.
10 48  BC          READ
10 49  11          0
10 4A  C3          JMP
10 4B  38          CHKFRM
10 4C  10          0

10 4D  CD  AIN,    CALL      /AN ADDRESS FLAG WAS FOUND
10 4E  58          BYTE      /THE NEXT 4 FRAMES REPRESENT
10 4F  10          0         /THE H&L ADDRESS
10 50  67          MOVHA
10 51  CD          CALL
10 52  58          BYTE
10 53  10          0
10 54  6F          MOVLA     /SAVE THE ADDRESS IN H&L
10 55  C3          JMP
10 56  47          NXTIN
10 57  10          0

                   /THE FORMAT OF THE PAPER TAPE IS, THE 2 MSB'S ARE
                   /PUNCHED OUT FIRST, THEN THE REMAINING 6 BITS
                   /ARE PUNCHED OUT.

10 58  CD  BYTE,   CALL      /GET THE 2 MSB'S
10 59  BC          READ
10 5A  11          0
10 5B  0F  BYTE1,  RRC       /ROTATE THEM INTO THE MSB
10 5C  0F          RRC
10 5D  4F          MOVCA     /AND SAVE THEM IN C
10 5E  CD          CALL      /GET THE NEXT PAPER TAPE FRAME
10 5F  BC          READ
10 60  11          0
10 61  81          ADDC      /ADD THE MSB'S TO THE LSB'S
10 62  C9          RET       /AND RETURN WITH THE 8 BIT # IN A.

10 63  FE  NOREAD, CPI
10 64  50          "P"       /P FOR PUNCH A PAPER TAPE
10 65  C2          JNZ       /NOT A P, SEE IF IT WAS
10 66  B7          NOPUN     /A VALID HEX NUMBER BEING
10 67  10          0         /USED TO SPECIFY AN ADDRESS.
10 68  CD  PUNCH,  CALL      /AFTER THE P WAS TYPED IN, PRINT
10 69  3D          CRLF      /A CR & LF.
10 6A  11          0
10 6B  CD          CALL      /GET THE BEGINNING OF FILE ADDRESS
10 6C  4E          TWOHEX
10 6D  11          0
10 6E  EB          XCHG      /PUT IT IN H&L
10 6F  CD          CALL
10 70  3D          CRLF
10 71  11          0
```

```
10 72  CD            CALL     /GET THE END OF FILE ADDRESS
10 73  4E            TWOHEX
10 74  11            0
10 75  7B            MOVAE    /CALCULATE THE NUMBER OF BYTES
10 76  95            SUBL     /TO BE PUNCHED OUT.
10 77  5F            MOVEA
10 78  7A            MOVAD
10 79  9C            SBBH
10 7A  57            MOVDA
10 7B  13            INXD     /AND SET THE RESULT TO 1 MORE.
10 7C  CD            CALL     /PUNCH OUT 10" OF LEADER
10 7D  9D            LDDR
10 7E  10            0
10 7F  3E            MVIA     /PUNCH OUT AN ADDRESS FLAG
10 80  40            100
10 81  CD            CALL
10 82  DE            TTYOUT
10 83  11            0
10 84  7C            MOVAH
10 85  CD            CALL     /PUNCH OUT THE BEGINNING HI ADDRESS
10 86  A9            BYTOUT   /AS A 2 & 6 BIT BYTE.
10 87  10            0
10 88  7D            MOVAL
10 89  CD            CALL     /PUNCH OUT THE BEGINNING LO ADDRESS
10 8A  A9            BYTOUT   /AS A 2 & 6 BIT BYTE.
10 8B  10            0
10 8C  7E   NXTOUT,  MOVAM    /THEN START PUNCHING OUT THE
10 8D  CD            CALL     /CONTENTS OF MEMORY ADDRESSED BY H&L
10 8E  A9            BYTOUT
10 8F  10            0
10 90  23            INXH
10 91  1B            DCXD     /ANY MORE DATA TO BE PUNCHED OUT ?
10 92  7A            MOVAD
10 93  B3            ORAE
10 94  C2            JNZ      /YES, GET THE NEXT MEMORY LOCATION
10 95  8C            NXTOUT   /NO, THEN PUNCH OUT TRAILER AND
10 96  10            0        /RETURN TO THE COMMAND DECODER
10 97  CD            CALL
10 98  9D            LDDR
10 99  10            0
10 9A  C3            JMP
10 9B  25            QUEST2
10 9C  10            0
10 9D  0E   LDDR,    MVIC
10 9E  64            144      /C=144=100 DECIMAL=10" OF LEADER
10 9F  3E            MVIA
10 A0  80            200      /CODE FOR LEADER AND TRAILER
10 A1  CD            CALL
10 A2  DE            TTYOUT   /PUNCH THE 200 CODE
10 A3  11            0
10 A4  0D            DCRC     /10" OF LEADER OR TRAILER YET ?
10 A5  C2            JNZ      /NO, PUNCH SOME MORE
10 A6  9F            LDDR+2
10 A7  10            0
```

```
10 A8  C9             RET        /YES, RETURN

10 A9  4F   BYTOUT,   MOVCA      /DIVIDE AN 8 BIT BYTE INTO
10 AA  E6             ANI        /A 2 BIT AND A 6 BIT BYTE
10 AB  C0             300
10 AC  07             RLC
10 AD  07             RLC
10 AE  CD             CALL       /PUNCH OUT THE 2 MSB'S
10 AF  DE             TTYOUT
10 B0  11             0
10 B1  79             MOVAC
10 B2  E6             ANI        /THEN PUNCH OUT THE 6 LSB'S
10 B3  3F             077
10 B4  C3             JMP
10 B5  DE             TTYOUT
10 B6  11             0

                      /A "K" TYPED IN MEANS TO REMOVE THE BREAKPOINT.
                      /A "X" TYPED IN MEANS TO CONTINUE EXECUTION
                      /A "S" TYPED IN MEANS TO SINGLE STEP

                      /SINCE THESE ARE COMMANDS WHICH REQUIRE THE
                      /USE OF THE BREAKPOINT, DBUG CHECKS TO SEE
                      /WHETHER THE USER HAS SET A BREAKPOINT
                      /SOMEPLACE. IF ONE HAS NOT BEEN SET, THEN
                      /WE CANNOT EXECUTE THESE COMMANDS.

10 B7  5F   NOPUN,    MOVEA      /SAVE THE CHARACTER IN E
10 B8  FE   BCS,      CPI        /S IS FOR SINGLE STEP
10 B9  53             "S"
10 BA  CA             JZ         /IT WAS A S, EXECUTE 1 INSTRUCTION !
10 BB  5C             STEP
10 BC  13             0
10 BD  FE             CPI        /X IS FOR CONTINUE (FULL SPEED)
10 BE  58             "X"        /PROGRAM EXECUTION
10 BF  CA             JZ         /IT WAS A X, CONTINUE EXECUTION
10 C0  3C             CONTIN
10 C1  13             0
10 C2  FE             CPI        /K IS FOR REMOVE THE LAST BREAKPOINT
10 C3  4B             "K"        /(THE USER HAS FOUND A BETTER PLACE)
10 C4  C2             JNZ        /NOT A "K"
10 C5  D9             NOBCS      /NOT S,K OR X, MAYBE ITS A HEX NUMBER
10 C6  10             0
10 C7  2A             LHLD       /A "K" WAS TYPED TO REMOVE A BREAKPOINT
10 C8  C0             BRKADD     /NOW SEE IF THERE IS ONE TO REMOVE !
10 C9  03             0
10 CA  7E             MOVAM      /IS THERE A BREAKPOINT
10 CB  FE             CPI        /IN R-W MEMORY ?
10 CC  FF             377
10 CD  C2             JNZ        /NO BREAKPOINT, THEREFORE THE S,K OR X
10 CE  1D             QUEST+1    /CANNOT BE USED
10 CF  10             0
10 D0  CD             CALL       /IT WAS A K, TAKE THE CONTENTS OF "TMP"
```

```
10 D1  7F            RSTRE      /AND PUT IT BACK IN THE USER'S PROGRAM
10 D2  13            0          /USING "BRKADD" AS THE ADDRESS.
10 D3  CD            CALL       /SINCE THE BREAKPOINT HAS BEEN REMOVED
10 D4  A0            ZEROIT     /WE HAVE TO RESET BRKADD TO 377 377
10 D5  13            0          /TO PREVENT THE USE OF K,X, OR S
10 D6  C3            JMP
10 D7  25            QUEST2     /NOW GET ANOTHER COMMAND.
10 D8  10            0
10 D9  CD   NOBCS,   CALL       /IT WAS A VALID HEX NUMBER, SO
10 DA  47            SPCHEX     /CONVERT THE ASCII TO HEX
10 DB  11            0          /(THERE MUST BE 3 DIGITS TYPED IN)
10 DC  53            MOVDE      /SAVE THE HI ADDRESS IN D
10 DD  CD            CALL       /NOW GET THE 3 DIGIT LO ADDRESS
10 DE  52            HEXIN
10 DF  11            0
10 E0  EB            XCHG       /THE ADDRESS IS NOW IN H&L
10 E1  CD            CALL       /NOW THAT AN ADDRESS HAS BEEN SPECIFIED,
10 E2  D9            TTYIN      /SEE WHAT THE USER WANTS TO DO AT
10 E3  11            0          /THAT ADDRESS !
10 E4  FE            CPI
10 E5  47            "G"        /G=GO, BEGINNING AT THE USER'S ADDRESS
10 E6  C2            JNZ        /A G WAS NOT TYPED IN, TRY A B
10 E7  F5            NOGO
10 E8  10            0
10 E9  CD            CALL       /PRINT OUT A CR, LF AFTER THE G
10 EA  3D            CRLF       /WAS TYPED IN
10 EB  11            0
10 EC  AF            XRAA       /SET A - E TO 000
10 ED  47            MOVBA
10 EE  4F            MOVCA
10 EF  57            MOVDA
10 F0  5F            MOVEA
10 F1  E5            PUSHH      /PUSH THE ADDRESS ON THE STACK
10 F2  67            MOVHA      /THEN SET H&L TO 000
10 F3  6F            MOVLA
10 F4  C9            RET        /POP THE USER'S ADDRESS INTO THE
                                /PROGRAM COUNTER, AND OFF YOU GO.


10 F5  FE   NOGO,    CPI        /B AFTER AN ADDRESS = SET A BREAKPOINT.
10 F6  42            "B"
10 F7  CA            JZ         /IT WAS A B, SO SET UP THE BREAKPOINT
10 F8  FE            BREAK
10 F9  11            0
10 FA  FE            CPI        /L=LIST THE CONTENTS OF MEMORY
10 FB  4C            "L"        /BEGINNING AT THE USER
10 FC  CA            JZ         /SPECIFIED ADDRESS.
10 FD  80            LIST
10 FE  11            0
10 FF  FE            CPI        /A "/" MEANS TYPE OUT THE
11 00  2F            "/"        /CONTENTS OF MEMORY.
11 01  C2            JNZ
11 02  1D            QUEST+1    /NOT A G,B,L OR / SO TYPE OUT
11 03  10            0          /A ? AND GET ANOTHER COMMAND.
```

```
11 04  CD            CALL     /WE TYPED A SLASH SO PUT OUT A SPACE
11 05  F2            SPC
11 06  11            0
11 07  7E  NXTLOC,   MOVAM    /GET THE CONTENTS OF MEMORY
11 08  CD            CALL
11 09  9C            HEXOUT   /PRINT IT OUT AS A 2
11 0A  11            0        /DIGIT HEX NUMBER
11 0B  CD            CALL     /DID THE USER THEN TYPE IN A
11 0C  CF            TTYI     /CR, LF OR A HEX # ?
11 0D  11            0
11 0E  E6            ANI
11 0F  7F            177
11 10  FE            CPI
11 11  0D            015      /CR=RETURN TO THE COMMAND DECODER
11 12  CA            JZ
11 13  25            QUEST2
11 14  10            0
11 15  FE            CPI
11 16  0A            012      /LF=CONTENTS OF THE NEXT MEMORY LOCATION
11 17  C2            JNZ      /ANOTHER HEX NUMBER WAS TYPED IN
11 18  24            CNT      /(NEW CONTENTS OF THE SAME LOCATION)
11 19  11            0
11 1A  CD  NXTLN,    CALL     /IT WAS A LF, SO PRINT OUT THE
11 1B  3D            CRLF
11 1C  11            0
11 1D  23            INXH
11 1E  CD            CALL     /ADDRESS AND CONTENTS OF THE NEXT
11 1F  EA            HLSLSH   /CONSECUTIVE MEMORY LOCATION
11 20  11            0        /HLSLSH=H&L THEN A / AND SPACE.
11 21  C3            JMP
11 22  07            NXTLOC   /NOW TYPE OUT THE CONTENTS OF
11 23  11            0        /THE NEXT LOCATION

11 24  CD  CNT,      CALL     /OUTPUT THE FIRST CHARACTER (NUMBER)
11 25  DE            TTYOUT
11 26  11            0
11 27  CD            CALL     /GO TO THE HEX INPUT ROUTINE
11 28  47            SPCHEX
11 29  11            0
11 2A  73            MOVME    /SAVE THE NUMBER INPUT
11 2B  CD            CALL     /SEE IF A CR, LF OR HEX
11 2C  CF            TTYI     /NUMBER WAS THEN TYPED IN
11 2D  11            0
11 2E  E6            ANI
11 2F  7F            177
11 30  FE            CPI
11 31  0D            015      /CR=RETURN TO THE COMMAND DECODER
11 32  CA            JZ
11 33  25            QUEST2
11 34  10            0
11 35  FE            CPI
11 36  0A            012
11 37  C2            JNZ      /IT WASN'T A LF, IT MUST BE
11 38  24            CNT      /NEW CONTENTS AGAIN !
```

```
11 39  11             0
11 3A  C3             JMP
11 3B  1A             NXTLN        /A LF WAS TYPED IN, TYPE OUT THE
11 3C  11             0            /NEXT MEM. LOCATION AND ITS CONTENTS

11 3D  3E   CRLF,     MVIA         /TYPE OUT A CR & LF
11 3E  8D             215
11 3F  CD             CALL
11 40  DE             TTYOUT
11 41  11             0
11 42  3E             MVIA
11 43  8A             212
11 44  C3             JMP
11 45  DE             TTYOUT
11 46  11             0

                      /THIS IS THE HEXADECIMAL INPUT ROUTINE

11 47  1E   SPCHEX,   MVIE         /E IS A TEMPORARY STORAGE POINTER
11 48  00             000
11 49  0E             MVIC         /C IS A DIGIT INPUT COUNTER
11 4A  02             002
11 4B  C3             JMP
11 4C  59             NXTHEX+3
11 4D  11             0

11 4E  CD   TWOHEX,   CALL         /GET ONE TWO DIGIT HEX NUMBER
11 4F  52             HEXIN
11 50  11             0
11 51  53             MOVDE        /SAVE THE VALUE IN "D"
11 52  AF   HEXIN,    XRAA
11 53  5F             MOVEA
11 54  0E             MVIC
11 55  02             002
11 56  CD   NXTHEX,   CALL         /GET A TTY CHARACTER
11 57  D9             TTYIN
11 58  11             0
11 59  FE             CPI          /AND SEE IF IT IS A VALID HEX #
11 5A  30             060          /IF IT ISN'T, TYPE.A ?
11 5B  DA             JC
11 5C  1C             QUEST
11 5D  10             0
11 5E  FE             CPI
11 5F  47             "G"
11 60  D2             JNC
11 61  1C             QUEST
11 62  10             0
11 63  FE             CPI
11 64  3A             ":"
11 65  DA             JC
11 66  6F             HEX09
11 67  11             0
11 68  FE             CPI
11 69  41             "A"
```

```
11 6A  DA            JC
11 6B  1C            QUEST
11 6C  10            0
11 6D  C6            ADI
11 6E  09            011
11 6F  E6  HEX09,    ANI
11 70  0F            017
11 71  47            MOVBA
11 72  7B            MOVAE
11 73  07            RLC
11 74  07            RLC
11 75  07            RLC
11 76  07            RLC
11 77  80            ADDB
11 78  5F            MOVEA
11 79  0D            DCRC
11 7A  C2            JNZ
11 7B  56            NXTHEX
11 7C  11            0
11 7D  C3            JMP
11 7E  F2            SPC
11 7F  11            0
                      /L=LIST THE CONSECUTIVE MEMORY LOCATIONS
                      /AND THEIR CONTENTS UNTIL THE USER PRESSES
                      /A PRINTING TELETYPE KEY.

11 80  CD  LIST,     CALL      /TYPE A CR & LF
11 81  3D            CRLF
11 82  11            0
11 83  CD            CALL      /THEN THE H&L ADDRESS, A SLASH
11 84  EA            HLSLSH    /AND A SPACE
11 85  11            0
11 86  7E            MOVAM     /GET THE CONTENTS OF MEMORY
11 87  CD            CALL      /AND PRINT IT OUT.
11 88  9C            HEXOUT
11 89  11            0
11 8A  23            INXH      /INCREMENT THE MEMORY POINTER
11 8B  DB            IN        /ANY TELETYPE KEY PRESSED YET ?
11 8C  11            021       /TTY SENSE REG.
11 8D  E6            ANI
11 8E  01            001
11 8F  CA            JZ        /NO, LIST OUT THE NEXT LOCATION
11 90  80            LIST
11 91  11            0
11 92  DB            IN        /YES, INPUT THE TERMINATING
11 93  10            020       /CHARACTER
11 94  C3            JMP
11 95  25            QUEST2    /AND GO TO THE COMMAND DECODER.
11 96  10            0

                      /TYPE OUT THE CONTENTS OF H&L AS 2
                      /2 DIGIT HEX NUMBERS, WITH A SPACE AFTER
                      /THE HI AND LO ADDRESS.
```

```
11 97  7C  HLOUT,   MOVAH
11 98  CD           CALL
11 99  9C           HEXOUT
11 9A  11           0
11 9B  7D           MOVAL
11 9C  CD  HEXOUT,  CALL
11 9D  A6           HEX
11 9E  11           0
11 9F  79           MOVAC
11 A0  CD           CALL
11 A1  A6           HEX
11 A2  11           0
11 A3  C3           JMP
11 A4  F2           SPC
11 A5  11           0

11 A6  0F  HEX,     RRC
11 A7  0F           RRC
11 A8  0F           RRC
11 A9  0F           RRC
11 AA  4F           MOVCA
11 AB  E6           ANI
11 AC  0F           017
11 AD  C6           ADI
11 AE  B0           260
11 AF  FE           CPI
11 B0  BA           272
11 B1  DA           JC
11 B2  B6           NMB09
11 B3  11           0
11 B4  C6           ADI
11 B5  07           007
11 B6  C3  NMB09,   JMP
11 B7  DE           TTYOUT
11 B8  11           0
11 B9  00           0
11 BA  00           0
11 BB  00           0

11 BC  D3  READ,    OUT        /PULSE THE READER CONTROL RELAY
11 BD  11           021
11 BE  00           NOP
11 BF  00           NOP
11 C0  00           NOP
11 C1  00           NOP
11 C2  00           NOP
11 C3  00           NOP
11 C4  00           NOP
11 C5  00           NOP
11 C6  00           NOP
11 C7  00           NOP
11 C8  00           NOP
11 C9  00           NOP
11 CA  00           NOP
```

```
11 CB 00            NOP
11 CC 00            NOP
11 CD 00            NOP
11 CE 00            NOP
11 CF DB  TTYI,     IN         /HAS A CHARACTER BEEN RECEIVED YET ?
11 D0 11            021
11 D1 E6            ANI
11 D2 01            001
11 D3 CA            JZ         /NO, KEEP WAITING FOR THE FLAG
11 D4 CF            TTYI
11 D5 11            0
11 D6 DB            IN         /YES, THEN INPUT IT
11 D7 10            020
11 D8 C9            RET

11 D9 CD  TTYIN,    CALL       /GET A TELETYPE CHARACTER
11 DA CF            TTYI
11 DB 11            0
11 DC E6            ANI        /AND THEN ECHO IT
11 DD 7F            177
11 DE 47  TTYOUT,   MOVBA      /SAVE THE CHARACTER TO BE PRINTED IN B
11 DF DB            IN         /IS THE TRANSMITTER READY YET ?
11 E0 11            021
11 E1 E6            ANI
11 E2 04            004
11 E3 CA            JZ         /NO, KEEP WAITING
11 E4 DF            TTYOUT+1
11 E5 11            0
11 E6 78            MOVAB      /YES, GET THE CHARACTER
11 E7 D3            OUT        /AND TRANSMIT IT.
11 E8 10            020
11 E9 C9            RET

11 EA CD  HLSLSH,   CALL
11 EB 97            HLOUT      /TYPE OUT H & L, THEN A SPACE
11 EC 11            0
11 ED 3E            MVIA       /THEN TYPE A "/"
11 EE AF            257
11 EF CD            CALL
11 F0 DE            TTYOUT
11 F1 11            0

                    /PRINT OUT SPACES, THE NUMBER OF WHICH
                    /IS STORED IN C.

11 F2 0E  SPC,      MVIC
11 F3 01            001
11 F4 3E  MORSPC,   MVIA
11 F5 A0            240        /ASCII SPACE
11 F6 CD            CALL
11 F7 DE            TTYOUT
11 F8 11            0
11 F9 0D            DCRC       /ANYMORE TO PRINT ?
11 FA C2            JNZ        /YES, PRINT ANOTHER 1
```

```
11 FB  F4            MORSPC
11 FC  11            0
11 FD  C9            RET       /NO, THEN RETURN

11 FE  CD  BREAK,    CALL      /THE USER WANTS TO PUT A BREAKPOINT
11 FF  04            BRK       /IN A PROGRAM
12 00  12            0
12 01  C3            JMP
12 02  25            QUEST2
12 03  10            0

12 04  22  BRK,      SHLD      /SAVE THE ADDRESS OF THE BREAK-
12 05  C0            BRKADD    /POINT ADDRESS IN BRKADD.
12 06  03            0
12 07  EB            XCHG      /NOW WE WANT TO DUPLICATE PART OF
12 08  21            LXIH      /USER'S PROGRAM, STARTING AT THE
12 09  C4            TMP       /BREAKPOINT ADDRESS, DOWN AT "TMP"
12 0A  03            0
12 0B  1A  BREAK1,   LDAXD
12 0C  47            MOVBA
12 0D  3E            MVIA      /WRITE A RST7 INTO THE USER'S PROGRAM
12 0E  FF            377       /THIS IS A RST7 INSTRUCTION
12 0F  12            STAXD
12 10  70            MOVMB
12 11  23            INXH
12 12  13            INXD
12 13  1A            LDAXD
12 14  77            MOVMA
12 15  23            INXH
12 16  13            INXD
12 17  1A            LDAXD
12 18  77            MOVMA
12 19  C9            RET       /WE HAVE DONE 3 MEMORY LOCATIONS

                     /WE "HIT" THE RST7 IN THE USER'S PROGRAM AND
                     /WE GET TO ENTRY BY THE JUMP INSTRUCTION STORED
                     /AT LOC. 00 38. THE RST7 MAY BE CHANGED
                     /AT THIS POINT, ALL OF THE REGISTERS
                     /AND THE STACK HAVE USER VALUES IN THEM
                     /SO DBUG HAS TO PRESERVE THEM.

12 1A  22  ENTRY,    SHLD      /SAVE H&L IN "TEMPO"
12 1B  CD            TEMPO
12 1C  03            0
12 1D  F5            PUSHPSW
12 1E  E1            POPH      /GET THE FLAGS AND A IN H&L
12 1F  22            SHLD      /NOW SAVE THEM IN "FLAG"
12 20  CF            FLAG
12 21  03            0
12 22  21            LXIH      /NOW DESTROY THE RETURN ON THE STACK
12 23  02            002       /DUE TO THE RST INSTRUCTION AND
12 24  00            000       /GET THE STACK POINTER IN H&L
12 25  39  SENTRY,   DADSP
12 26  22            SHLD      /SAVE THE USER'S STACK IN "USERSK"
```

```
12 27  CB      USERSK
12 28  03      0
12 29  31      LXISP     /NOW SET THE SP SO DBUG CAN USE IT
12 2A  FE      STACK
12 2B  03      0
12 2C  2A      LHLD      /GET H&L BACK INTO H&L FROM "TEMPO"
12 2D  CD      TEMPO
12 2E  03      0
12 2F  E5      PUSHH     /PUSH B - H ONTO DBUG'S STACK
12 30  D5      PUSHD
12 31  C5      PUSHB
12 32  2A      LHLD      /NOW GET THE FLAGS AND A INTO H&L
12 33  CF      FLAG
12 34  03      0
12 35  E5      PUSHH     /AND PUSH THEM ON THE STACK.
12 36  2A      LHLD      /GET THE USER SELECTED BREAKPOINT
12 37  C0      BRKADD    /ADDRESS AND TYPE IT OUT SO WE DON'T
12 38  03      0         /FORGET WHAT IT WAS.
12 39  CD      CALL
12 3A  97      HLOUT
12 3B  11      0
12 3C  3A      LDA       /GET THE USERS INSTRUCTION AT "TMP"
12 3D  C4      TMP
12 3E  03      0
12 3F  47      MOVBA     /AND SEE IF IT IS EXECUTABLE.
12 40  FE      CPI
12 41  C3      303       /CAN'T DO A JUMP
12 42  CA      JZ
12 43  A7      NOGOOD
12 44  12      0
12 45  FE      CPI
12 46  CD      315       /CAN'T CALL ANY SUBROUTINES
12 47  CA      JZ
12 48  A7      NOGOOD
12 49  12      0
12 4A  FE      CPI
12 4B  C9      311       /RETURN
12 4C  CA      JZ
12 4D  A7      NOGOOD
12 4E  12      0
12 4F  FE      CPI
12 50  E9      351       /PCHL
12 51  CA      JZ
12 52  A7      NOGOOD
12 53  12      0
12 54  FE      CPI
12 55  DB      333       /OUTPUT INSTRUCTIONS ARE 2 BYTE INSTR.
12 56  CA      JZ
12 57  9D      IMMED
12 58  12      0
12 59  FE      CPI
12 5A  D3      323       /AND SO ARE INPUT INSTRUCTIONS.
12 5B  CA      JZ
12 5C  9D      IMMED
```

```
12 5D  12          0
12 5E  E6  NO300,  ANI     /IT WASN'T 1 OF THOSE, TRY THESE.
12 5F  C7          307
12 60  FE          CPI
12 61  C6          306
12 62  CA          JZ
12 63  9D          IMMED
12 64  12          0
12 65  FE          CPI
12 66  C2          302     /CONDITIONAL JUMPS
12 67  CA          JZ
12 68  A7          NOGOOD
12 69  12          0
12 6A  FE          CPI
12 6B  C4          304     /CONDITIONAL CALLS
12 6C  CA          JZ
12 6D  A7          NOGOOD
12 6E  12          0
12 6F  FE          CPI
12 70  C0          300     /CONDITIONAL RETURNS
12 71  CA          JZ
12 72  A7          NOGOOD
12 73  12          0
12 74  FE          CPI
12 75  C7          307     /RESTARTS
12 76  CA          JZ
12 77  A7          NOGOOD
12 78  12          0
12 79  FE          CPI
12 7A  06          006     /IMMEDIATE MOVES
12 7B  CA          JZ
12 7C  9D          IMMED
12 7D  12          0
12 7E  FE          CPI
12 7F  01          001     /LOAD IMMEDIATE REGISTER PAIR
12 80  CA          JZ
12 81  94          IMMSL1
12 82  12          0
12 83  FE          CPI
12 84  02          002     /STA, LDA, SHLD OR LHLD
12 85  CA          JZ
12 86  B3          IMMSL2
12 87  12          0
12 88  21  SING,   LXIH    /IT WAS A SINGLE BYTE INSTRUCTION
12 89  00          000     /SO WE WRITE 2 NOPS IMMEDIATELY
12 8A  00          000     /AFTER THE INSTRUCTION.
12 8B  22          SHLD
12 8C  C5          TMP+1
12 8D  03          0
12 8E  2A          LHLD    /SET H&L = TO THE CONTENTS OF "BRKADD"
12 8F  C0          BRKADD
12 90  03          0
12 91  C3          JMP
12 92  BE          ONEMOR
```

```
12 93  12            0
12 94  7B  IMMSL1,   MOVAB     /IS THE INSTRUCTION REALLY
12 95  E6            ANI       /A SINGLE BYTE OR 3 BYTE ?
12 96  08            010
12 97  C2            JNZ       /IF A=010, ITS A SINGLE BYTE
12 98  88            SING
12 99  12            0
12 9A  C3            JMP       /IF A=000, ITS A 3 BYTE
12 9B  B9            THREBY
12 9C  12            0
12 9D  AF  IMMED,    XRAA      /IMMEDIATE AND I-O INSTRUCTIONS
12 9E  32            STA       /ARE TWO BYTE INSTRUCTIONS
12 9F  C6            TMP+2     /SO WE SAVE 1 NOP AFTER THE
12 A0  03            0         /INSTRUCTION.
12 A1  2A            LHLD
12 A2  C0            BRKADD
12 A3  03            0
12 A4  C3            JMP
12 A5  BD            TWOMOR
12 A6  12            0

12 A7  2A  NOGOOD,   LHLD      /IF THE BREAKPOINT WAS PLACED ON A
12 A8  C0            BRKADD    /NONEXECUTABLE INSTRUCTION, WE RE-
12 A9  03            0         /MOVE THE RST7 FROM THE USER'S
12 AA  22            SHLD      /PROGRAM AND GO TO THE COMMAND DECODER
12 AB  C2            NEWADD
12 AC  03            0
12 AD  CD            CALL
12 AE  7F            RSTRE
12 AF  13            0
12 B0  C3            JMP
12 B1  20            QUEST+4
12 B2  10            0
12 B3  78  IMMSL2,   MOVAB     /IS IT REALLY A STA,LDA,SHLD OR LHLD ?
12 B4  FE            CPI
12 B5  22            042
12 B6  DA            JC        /NO, THEN IT MUST BE A SINGLE BYTE.
12 B7  88            SING
12 B8  12            0
12 B9  2A  THREBY,   LHLD
12 BA  C0            BRKADD
12 BB  03            0
12 BC  23            INXH
12 BD  23  TWOMOR,   INXH
12 BE  23  ONEMOR,   INXH
12 BF  22            SHLD      /STORE THE ADDRESS OF THE NEXT
12 C0  C2            NEWADD    /INSTRUCTION
12 C1  03            0
12 C2  F1            POPPSW    /GET BACK ALL THE USER'S REGISTERS
12 C3  C1            POPB
12 C4  D1            POPD
12 C5  E1            POPH
12 C6  22            SHLD      /SAVE H&L IN "TEMPO"
12 C7  CD            TEMPO
```

```
12 C8  03             0
12 C9  2A             LHLD      /SET H&L WITH THE USER'S STACK POINTER
12 CA  CB             USERSK
12 CB  03             0
12 CC  F9             SPHL      /NOW SET THE SP WITH THAT VALUE
12 CD  2A             LHLD      /NOW GET H&L BACK TO WHAT THEY WERE
12 CE  CD             TEMPO
12 CF  03             0
12 D0  C3             JMP       /NOW DO THE 1,2 OR 3 BYTE INSTRUCTION
12 D1  C4             TMP       /STORED AT "TMP" (WE MAY ALSO DO
12 D2  03             0         /1 OR 2 NOP'S).

                      /AFTER EXECUTING THE INSTRUCTION AT "TMP"
                      /WE JUMP BACK TO HERE. THE CONTENTS OF
                      /THE REGISTERS MUST BE SAVED AGAIN, SP'S
                      /SWITCHED ETC. BEFORE WE CAN TYPE OUT THE
                      /CONTENTS OF THE REGISTERS.

12 D3  22    EXECUT,  SHLD      /FROM "TMP" WE JUMP TO HERE.
12 D4  CD             TEMPO     /SAVE H&L IN "TEMPO"
12 D5  03             0
12 D6  F5             PUSHPSW
12 D7  E1             POPH
12 D8  22             SHLD      /SAVE THE FLAGS AND A IN "FLAG"
12 D9  CF             FLAG
12 DA  03             0
12 DB  21             LXIH      /SET H&L = TO 000
12 DC  00             0
12 DD  00             0
12 DE  39             DADSP     /PUT THE USER'S SP INTO H&L
12 DF  22             SHLD      /AND SAVE IT IN  USERSK"
12 E0  CB             USERSK
12 E1  03             0
12 E2  31             LXISP     /SET UP A SP FOR DBUG TO USE
12 E3  FE             STACK
12 E4  03             0
12 E5  2A             LHLD      /SET H&L TO WHAT THEY WHERE
12 E6  CD             TEMPO
12 E7  03             0
12 E8  E5             PUSHH     /SAVE B-H ON THE STACK
12 E9  D5             PUSHD
12 EA  C5             PUSHB
12 EB  2A             LHLD      /GET THE FLAGS AND A INTO H&L
12 EC  CF             FLAG
12 ED  03             0
12 EE  E5             PUSHH     /AND PUT THEM ON THE STACK ALSO.
12 EF  CD             CALL      /TYPE OUT A CR & LF
12 F0  3D             CRLF
12 F1  11             0
12 F2  21             LXIH      /DECREMENT THE REGISTER LABEL COUNTER
12 F3  CA             LAB
12 F4  03             0
12 F5  35             DCRM
12 F6  CC             CZ        /IF 0, TYPE OUT THE REGISTER LABELS
```

```
12 F7  B2              LABEL
12 F8  13              0
12 F9  21              LXIH
12 FA  00              000
12 FB  00              000
12 FC  39              DADSP   /SET H&L = TO THE SP
12 FD  5E              MOVEM   /GET THE FLAG WORD
12 FE  CD              CALL    /AND DISSASSEMBLE IT INTO 1'S & 0'S
12 FF  87              BIT
13 00  13              0
13 01  23              INXH
13 02  7E              MOVAM   /NOW GET THE A REGISTER
13 03  CD              CALL    /AND TYPE IT OUT AS A 2 DIGIT HEX #
13 04  9C              HEXOUT
13 05  11              0
13 06  16              MVID    /NOW WE GET THE OTHER REGISTERS
13 07  03              003     /AND TYPE OUT THEIR HEX VALUES
13 08  1E    TWOTIM,   MVIE
13 09  02              002
13 0A  23              INXH
13 0B  23              INXH
13 0C  7E    MORI,     MOVAM
13 0D  CD              CALL
13 0E  9C              HEXOUT
13 0F  11              0
13 10  2B              DCXH
13 11  1D              DCRE
13 12  C2              JNZ
13 13  0C              MOR1
13 14  13              0
13 15  23              INXH
13 16  23              INXH
13 17  15              DCRD    /ANYMORE REGISTER PAIRS ?
13 18  C2              JNZ     /YES, DO ANOTHER
13 19  08              TWOTIM
13 1A  13              0
13 1B  56              MOVDM   /NO, GET THE VALUES OF H&L INTO H&L
13 1C  2B              DCXH
13 1D  5E              MOVEM
13 1E  EB              XCHG
13 1F  7E              MOVAM   /GET CNTS OF MEMORY ADDRESSED BY H & L
13 20  CD              CALL    /AND TYPE OUT ITS VALUE
13 21  9C              HEXOUT
13 22  11              0
13 23  2A              LHLD    /SET H&L = TO THE USER'S SP
13 24  CB              USERSK
13 25  03              0
13 26  CD              CALL    /AND TYPE OUT THE 2 HEX WORDS
13 27  97              HLOUT
13 28  11              0
13 29  23              INXH    /GET THE HI OFF THE STACK
13 2A  7E              MOVAM
13 2B  CD              CALL    /AND PRINT IT OUT
13 2C  9C              HEXOUT
```

```
13 2D  11             0
13 2E  2B             DCXH
13 2F  7E             MOVAM    /THEN GET THE LO OFF THE STACK
13 30  CD             CALL     /AND PRINT IT OUT
13 31  9C             HEXOUT
13 32  11             0
13 33  CD             CALL     /PRINT A CR & LF AFTER ALL OF THIS
13 34  3D             CRLF
13 35  11             0
13 36  CD             CALL     /REMOVE THE BREAKPOINT (RST7)
13 37  7F             RSTRE    /FROM THE USER'S PROGRAM
13 38  13             0
13 39  C3             JMP      /AND GO BACK TO THE COMMAND DECODER
13 3A  25             QUEST2
13 3B  10             0

                      /IF WE TYPE A X TO CONTINUE, WE HAVE TO GET ALL
                      /OF THE REGISTERS BACK THE WAY THEY WERE, WITH
                      /THE VALUES IN THEM THAT THE USER ESTABLISHED, AL-
                      /ONG WITH THE USER'S STACK POINTER, AND
                      /THEN WE CAN USE THE CONTENTS OF "NEWADD"
                      /AS THE ADDRESS FROM WHERE WE SHOULD CONTINUE THE
                      /PROGRAMS EXECUTION.

13 3C  2A    CONTIN,  LHLD
13 3D  C0             BRKADD   /IS THERE A BREAKPOINT ADDRESS ?
13 3E  03             0
13 3F  23             INXH
13 40  7C             MOVAH
13 41  B5             ORAL
13 42  CA             JZ       /NO, BRKADD WAS SET TO 377 377
13 43  ID             QUEST+1
13 44  10             0
13 45  CD             CALL     /THERE IS AN ADDRESS, CONTINUE
13 46  3D             CRLF
13 47  11             0
13 48  F1    CNTIN1,  POPPSW   /GET BACK A-H
13 49  C1             POPB
13 4A  D1             POPD
13 4B  E1             POPH
13 4C  22             SHLD     /SAVE H&L IN "TEMPO"
13 4D  CD             TEMPO
13 4E  03             0
13 4F  2A             LHLD     /GET THE USER'S SP
13 50  CB             USERSK
13 51  03             0
13 52  F9             SPHL     /AND PUT IT INTO THE SP
13 53  2A             LHLD     /GET H&L BACK THE WAY THEY SHOULD BE
13 54  CD             TEMPO
13 55  03             0
13 56  E5             PUSHH    /PUT H&L ON THE STACK
13 57  2A             LHLD     /GET THE ADDRESS WHERE WE SHOULD
13 58  C2             NEWADD   /CONTINUE PROGRAM EXECUTION
13 59  03             0
```

```
13 5A  E3          XTHL      /NEW ADDRESS ON STACK, H&L THE
13 5B  C9          RET       /WAY THEY WERE. HERE WE GO . . . .

                     /IF WE TYPE S FOR SINGLE STEP, WE WANT TO EXECUTE
                     /1 INSTRUCTION AND THEN SEE WHAT EFFECT IT HAD
                     /ON THE REGISTERS, STACK POINTER, STACK OR MEMORY.

13 5C  2A   STEP,  LHLD      /SEE IF THERE IS A BREAKPOINT
13 5D  C0          BRKADD    /ALREADY SET.
13 5E  03          0
13 5F  23          INXH
13 60  7C          MOVAH
13 61  B5          ORAL
13 62  CA          JZ        /IF NOT  BREAKPOINT, H=L=377
13 63  1D          QUEST+1   /AFTER INXH, H=L=000 AND WE ERROR
13 64  10          0
13 65  CD          CALL      /TYPE A SPACE AFTER THE S
13 66  F2          SPC
13 67  11          0
13 68  2A          LHLD      /GET THE ADDRESS OF THE NEXT EX-
13 69  C2          NEWADD    /ECUTABLE INSTRUCTION.
13 6A  03          0
13 6B  CD          CALL      /SET A BREAKPOINT AT THIS NEW ADDRESS
13 6C  04          BRK
13 6D  12          0
13 6E  F1          POPPSW    /GET THE REGISTERS AND SP BACK
13 6F  C1          POPB      /TO WHAT THEY SHOULD BE
13 70  D1          POPD
13 71  E1          POPH
13 72  22          SHLD
13 73  CD          TEMPO
13 74  03          0
13 75  2A          LHLD
13 76  CB          USERSK
13 77  03          0
13 78  F9          SPHL
13 79  21          LXIH
13 7A  00          0
13 7B  00          0
13 7C  C3          JMP       /THEN JUMP TO THE BEGINNING OF
13 7D  25          SENTRY    /BREAKPOINT ROUTINE
13 7E  12          0

13 7F  3A   RSTRE, LDA       /GET THE FIRST BYTE OF THE INSTRUCTION
13 80  C4          TMP
13 81  03          0
13 82  2A          LHLD      /GET THE ADDRESS OF WHERE IT CAME FROM
13 83  C0          BRKADD
13 84  03          0
13 85  77          MOVMA     /PUT IT BACK IN THE USER'S PROGRAM
13 86  C9          RET       /WRITING OVER THE RST7 INSTRUCTION !

                     /THIS SUBROUTINE IS USED TO UNPACK THE FLAG
                     /REGISTER INTO A STRING OF 8 1'S AND 0'S.
```

```
                        /ENTER WITH THE WORD TO BE UNPACKED IN E

13 87  0E  BIT,         MVIC        /C= THE ROTATE OR BIT COUNTER
13 88  08               010
13 89  7B  NXTBIT,      MOVAE       /GET THE WORD
13 8A  07               RLC         /ROTATE 1 BIT LEFT INTO THE CARRY
13 8B  5F               MOVEA       /AND SAVE THE WORD AGAIN
13 8C  DA               JC          /IF THE CARRY IS SET, WE ROTATED
13 8D  9B               ONE         /A 1 INTO IT, SO WE PRINT A 1.
13 8E  13               0
13 8F  3E               MVIA        /OTHERWISE, WE TYPE A 0
13 90  B0               260
13 91  CD  ONEOUT,      CALL        /PRINT THE CHARACTER
13 92  DE               TTYOUT
13 93  11               0
13 94  0D               DCRC        /8 BITS YET ?
13 95  C2               JNZ         /NO, DO ANOTHER ONE
13 96  89               NXTBIT
13 97  13               0
13 98  C3               JMP         /YES, NOW PRINT A SPACE
13 99  F2               SPC
13 9A  11               0
13 9B  3E  ONE,         MVIA
13 9C  B1               261
13 9D  C3               JMP
13 9E  91               ONEOUT
13 9F  13               0

                        /THIS CLEARS THE CONTENTS OF "BRKADD"

13 A0  21  ZEROIT,      LXIH
13 A1  FF               377         /SET BRKADD TO 377 377
13 A2  FF               377
13 A3  22               SHLD
13 A4  C0               BRKADD
13 A5  03               0
13 A6  C9               RET

                        /THIS DUPLICATES PORTIONS OF "BUFF2"
                        /INTO R-W MEMORY

13 A7  0E  SETUP,       MVIC
13 A8  03               003
13 A9  7E               MOVAM
13 AA  12               STAXD
13 AB  23               INXH
13 AC  13               INXD
13 AD  0D               DCRC
13 AE  C2               JNZ
13 AF  A9               SETUP+2
13 B0  13               0
13 B1  C9               RET
```

```
                        /IF WE HAD DECREMENTED "LAB" TO 0, WE COME HERE.
                        /WE USE THE CONTENTS OF "REGLST"
                        /TO PRODUCE THE REGISTER LABELS. NOTICE THAT ANY
                        /NUMBER IN "REGLST" LESS THAN 010 (OCTAL), PRODUCES
                        /THAT NUMBER OF SPACES, EXCEPT 0, WHICH TERMINATES
                        /THE OUTPUT.

13 B2  36  LABEL,   MVIM     /SET THE LABEL COUNTER TO 5 AGAIN
13 B3  05           005
13 B4  21           LXIH
13 B5  CE           REGLST   /H&L POINT TO "REGLST"
13 B6  13           0
13 B7  7E  NXTLET,  MOVAM    /GET A "REGLST" TABLE CHARACTER
13 B8  FE           CPI      /IS IT A 000 ? (END OF THE TABLE ?)
13 B9  00           0
13 BA  C8           RZ       /YES, THEN RETURN
13 BB  FE           CPI      /IS IT A NUMBER BELOW 010 ?
13 BC  08           010      /IF SO, USE IT AS A SPACE COUNT
13 BD  DA           JC       /YES, SO TYPE SOME SPACES
13 BE  C7           SPCIT
13 BF  13           0
13 C0  CD           CALL     /NONE OF THE ABOVE, JUST
13 C1  DE           TTYOUT   /PRINT THE CHARACTER.
13 C2  11           0
13 C3  23  NXTLAB,  INXH     /INCREMENT REGISTER PAIR H&L
13 C4  C3           JMP      /THEN GET AND INTERPRETE THE NEXT CHARACTER
13 C5  B7           NXTLET
13 C6  13           0
13 C7  4F  SPCIT,   MOVCA
13 C8  CD           CALL
13 C9  F4           MORSPC
13 CA  11           0
13 CB  C3           JMP
13 CC  C3           NXTLAB
13 CD  13           0
13 CE  D3  REGLST,  323      /S
13 CF  DA           332      /Z
13 D0  01           001      /1 SPACE
13 D1  B1           261      /1
13 D2  01           001      /1 SPACE
13 D3  D0           320      /P
13 D4  01           001      /1 SPACE
13 D5  B2           262      /2 (4 BIT CARRY)
13 D6  01           001      /1 SPACE
13 D7  C1           301      /A
13 D8  02           002      /2 SPACES
13 D9  C2           302      /B
13 DA  02           002      /2 SPACES
13 DB  C3           303      /C
13 DC  02           002      /2 SPACES
13 DD  C4           304      /D
13 DE  02           002      /2 SPACES
13 DF  C5           305      /E
13 E0  02           002      /2 SPACES
```

```
023 341 310           310      /H
023 342 003           003      /3 SPACES
023 343 314           314      /L
023 344 003           003      /3 SPACES
023 345 315           315      /M
023 346 004           004      /4 SPACES
023 347 323           323      /S
023 350 001           001      /1 SPACE
023 351 320           320      /P
023 352 005           005      /5 SPACES
023 353 303           303      /C
023 354 001           001      /1 SPACE
023 355 323           323      /S
023 356 215           215      /CR
023 357 212           212      /LF
023 360 000           000      /MESSAGE TERMINATOR
023 361 303  BUFF2,   JMP      /THIS IS WHAT GETS DUPLICATED
023 362 032           ENTRY    /IN RAM. THIS STARTS AT 000 070
023 363 022           0
023 364 303           JMP      /THIS GOES TO 003 307
023 365 323           EXECUT
023 366 022           0
023 367 001           001      /THIS IS THE VALUE OF "LAB"
023 370 200           200
023 371 003           003
```

```
AIN     = 10 4D  BRKADD = 03 C0  BYTE   = 10 58  BYTE1  = 10 5B
BYTOUT  = 10 A9  BCS    = 10 B8  BREAK  = 11 FE  BRK    = 12 04
BREAK1  = 12 0B  BIT    = 13 87  BUFF2  = 13 F1  CHKFRM = 10 38
CNT     = 11 24  CRLF   = 11 3D  CONTIN = 13 3C  CNTIN1 = 13 48
ENTRY   = 12 1A  EXECUT = 12 D3  FLAG   = 03 CF  HEXIN  = 11 52
HEX09   = 11 6F  HLOUT  = 11 97  HEXOUT = 11 9C  HEX    = 11 A6
HLSLSH  = 11 EA  IMMSL1 = 12 94  IMMED  = 12 9D  IMMSL2 = 12 B3
LAB     = 03 CA  LDDR   = 10 9D  LIST   = 11 80  LABEL  = 13 B2
MORSPC  = 11 F4  MOR1   = 13 0C  NEWADD = 03 C2  NOEND  = 10 42
NXTIN   = 10 47  NOREAD = 10 63  NXTOUT = 10 8C  NOPUN  = 10 B7
NOBCS   = 10 D9  NOGO   = 10 F5  NXTLOC = 11 07  NXTLN  = 11 1A
NXTHEX  = 11 56  NMB09  = 11 B6  NO300  = 12 5E  NOGOOD = 12 A7
NXTBIT  = 13 89  NXTLET = 13 B7  NXTLAB = 13 C3  ONEMOR = 12 BE
ONEOUT  = 13 91  ONE    = 13 9B  PUNCH  = 10 68  QUEST  = 10 1C
QUEST2  = 10 25  QUEST3 = 10 28  RDR    = 10 30  READ   = 11 BC
RSTRE   = 13 7F  REGLST = 13 CE  STACK  = 03 FE  START  = 10 00
SPCHEX  = 11 47  SPC    = 11 F2  SENTRY = 12 25  SING   = 12 88
STEP    = 13 5C  SETUP  = 13 A7  SPCIT  = 13 C7  TMP    = 03 C4
TEMPO   = 03 CD  TWOHEX = 11 4E  TTY1   = 11 CF  TTYIN  = 11 D9
TTYOUT  = 11 DE  THREBY = 12 B9  TWOMOR = 12 BD  TWOTIM = 13 08
USERSK  = 03 CB  VECT   = 00 38  ZEROIT = 13 A0
ERRORS DETECTED = 000
```

L. 6.000

Il **dr. Christopher A. Titus** lavora per la Tychon in qualità di ingegnere sull'applicazione dei microcomputer Si è laureato in Fisica al Virginia Polytechnic Institute lavorando sulla strumentazione chimica automatizzata a microcomputer È coautore di numerosi articoli sulla strumentazione e ha tenuto numerose relazioni durante le più importanti conferenze scientifiche e di ingegneria negli USA, Chris ha programmato i microprocessori 8008, 8080 e 6502. Ha creato editor, assembler, disassembler e software per il debug e completi sistemi operativi per microcomputer. Inoltre è esperto di programmazione PDP-8 e progettazione digitale. Dal 1973 si interessa delle aree applicative dei computer. Attualmente cura soprattutto sistemi software, sistemi di acquisizioni dati e problematiche hardware e software.

Mr. Jonathan A. Titus è presidente della Tychon Inc La maggior parte della sua attività alla Tychon è concentrata sulle applicazioni di microcomputer e sui piccoli sistemi a microcomputer. Ha scritto ed ha collaborato alla realizzazione di articoli sulla strumentazione e sui microcomputer, pubblicati su riviste sia a carattere professionale che hobbistico. Jon iniziò ad interessarsi ai microcomputer nel '71 quando lavorava all'automazione di strumenti per analisi chimica La sua prima esperienza fu il microcomputer MARK 8, che era basato sull'8008. I suoi interessi sono ora rivolti all'8080 e ai microcomputer a 16 bit. Ha tenuto corsi per la American Chemical Society, attualmente programma corsi di Hardware e Software per la Tychon.

28

DBUG

Un programma interprete per
la messa a punto del software 8080

C. A. TITUS
J. A. TITUS

JACKSON ITALIANA EDITRICE